Kate

The *Keen Android Travel Extension*

" Rien ne sert de courir; il faut partir à point."

Jean Fontaine in Fables, 1668

(running won't do the trick, just leave at the right time)

Delft University of Technology
Faculty EEMCS

June 23, 2011

Bachelor Project group (IN3405)
Loubna Aammari, 1550756
Marco Hazebroek, 9313504
Jozua Sijsling, 1308319
Randy Tjin-Asjoe, 1304593

Thesis Committee
Chairman: Prof. Dr. Drs. L.J.M. Rothkrantz
B.Sc. Coordinator: Drs. P.R. van Nieuwenhuizen
Company Supervisor: Ir. P. van den Haak

TNO innovation for life

TU Delft Delft University of Technology

Preface

To complete the Bachelor education, students at TU Delft have to execute a complete software development project in the final quarter of year three. This is a project for 15 ects (420 hours), in which students have to perform an external internship in a project group, consisting of four persons. This project group has chosen a project at TNO (a major innovation organization in the Netherlands) in the spring and the summer of 2011. The group was bound by a common interest in the Android platform and the belief that this platform will see an substantial growth in the near future, both in market share and in functionalities. At the same time, TNO was looking for students to help them gain more knowledge of Android app making, as a means of supporting their vast traffic management knowledge.

This is the final report (IN3405) for the Bachelor of Science education in Computer Science for the section Man Machine Interaction, department Mediamatics, faculty Electrical Engineering, Mathematics and Computer Science (EEMCS).

The complete project documentation has been included - in Dutch - as appendices of this report. In Chapter 1, table 1, these documents are explained.

This Thesis consists of nine chapters.

Chapter 1 describes the context of the problem and requirements to possible solutions.
In chapter 2, related work and results of the research is covered.
Chapter 3 is a summary of the used tools and methods in this project.
In chapter 4, the transformation from problem to solution is visualized with the architecture of the app.
Chapter 5 describes the implementation phase.
Testing is crucial in every software engineering project, this subject is explained in chapter 6.
The conclusion is given in chapter 7, including a list of future work.
References and a list of definitions can be found in chapter 8 and 9, respectively.

We have realized this project with great enthusiasm and we hope that you will enjoy reading about it in this report. No specific knowledge about the Android platform is needed to comprehend this report. The font used for this report is *Droid Serif*, the obvious choice for reports about Android.

Delft, June 2011

Loubna Aammari	*Loubna.Aammari@gmail.com*
Marco Hazebroek	*Marco.Hazebroek@gmail.com*
Jozua Sijsling	*Jozua.Sijsling@gmail.com*
Randy Tjin-Asjoe	*RTjinasjoe@gmail.com*

Acknowledgements

Group: This thesis is the final Bachelor result for the project group. It is the summary of a three-year study, although some of us did not complete it within three years. This project was completed in cooperation with TNO, department Mobility. Gratitude to Bjorn Heijligers, project leader at TNO. Above all, we would like to thank Leon Rothkrantz (TU Delft) and Paul van den Haak (TNO) for their help and guidance in this project. Without them, Kate would never exist.

Loubna: These last couple of months have been exceptionally difficult for me. First with my sick younger brother and then, in the last phase of this project, getting ill myself. Would it not have been for the understanding and support of these amazing group members, I would probably have given up halfway. Jozua, Randy and Marco, much gratitude for pulling me through this. We were a rather unconventional group, but we made it work.
Ayyoub, little brother, thank you for being so strong. I hope you come home soon so you can meet Kate.

Marco: Finally, after all these years, I can close this chapter in my life. Thank you Loubna, Jozua en Randy, for your optimism and understanding, I really needed it in some dark periods. I guess our cooperation was meant to be. Big thanks to my mother and my mother-in-law for the never-ending support. Thank you Marvin, Gavin and Caithlin, you're such wonderful kids! Without you, this would never have happened. Monique, thank you for 21 years of support, of believing in me, I love you, this is all for you.

Jozua: Thanks go out to Randy, Loubna and Marco. They are good teammates, much more than I had anticipated. I wasn't very excited to go into this project at first. I did not like the constraints of full-time work for no-pay at all. On top of that I'd be working with a team of students I didn't know at all. Fortunately, our team ended up being very capable and very flexible. Marco especially was able to spark some of my leadership qualities, his enormous enthusiasm was rather contagious and I really have to thank him for that. Loubna and Randy, great work on the code, you've surprised me at times and I'm really glad to say that helped me appreciate the team even more.

Randy: It has taken me longer than expected to finally reach this part of my bachelor study. This could not have been possible without the loving support of my parents who have always been there in my time of need. A special thanks to my friend Erno who has been very encouraging throughout these past years. Of course last but not least my thanks to Jozua, Marco and Loubna for working together into making this a fruitful and memorable project.

Title: Building Kate, a Bachelor project (IN3405) at TNO, June 2011

Students:
Loubna Aammari, 1550756
Marco Hazebroek, 9313504
Jozua Sijsling, 1308319
Randy Tjin-Asjoe, 1304593

Thesis Committee, Delft University of Technology:
Chairman: Prof. Dr. Drs. L.J.M. Rothkrantz
BSc. Coordinator: Drs. P.R. van Nieuwenhuizen
Company Supervisor: Ir. P. van den Haak

Abstract

Road congestion causes a lot of problems for the Netherlands. Time and money is lost due to the high traffic load on Dutch highways. The unexpected extra travel time will cause travellers to arrive late at appointments. There are several strategies to reduce road congestion, one of which is the use of IT. The Dutch government is investing money to provide good travel information to every traveller in the year 2015, anytime, anyplace, so that the traveller can make an optimal and conscious choice, based on actual and reliable, nationwide, door-to-door travel information, both pre-trip and on-trip. The government cooperates with TNO, the Netherlands Organization for Applied Scientific Research. TNO has a vast knowledge of traffic management. TNO researches Intelligent Travel Systems to benefit both the individual traveller and the collective. Mobile devices will become increasingly more important as platform for such systems. The department Mobility wants a proof of concept that the mobile Android platform can act as a personal travel assistant to its user.

The project group has made Kate, the Keen Android Travel Extension. This Java-based Android app can determine the journeys to be made by the user today and tomorrow by reading appointments from the Android Calendar. Kate is a personal digital assistant. When the user is busy and has no time to search for travel information, Kate will already present it. Kate uses an external Travel Time Predictor (TTP) with access to historical and actual traffic data to determine the most probable travelling time. The TTP is accessed via a BackOffice Server at TNO. The same server is used to log all travel data, from which TNO can gather vital information about travel behavior and delays. So, the user gets predictions for free and in return sends back travel data to improve those predictions. That can help to improve predictions from TTP's, so the collective will benefit from the actions of the individual.

Kate will calculate the ideal departure time and will warn the user several times when leaving is almost imminent. The app can generate weekly and monthly reports about the journeys made by the user.

Kate is a working prototype that shows that an app can assist a traveller in the travel decision process. Kate is built up from modules, the source of travel data (now the calendar) and travel time prediction (now Tripcast from Model IT) can easily be switched to another source which will require only the change of one module. Adding a new language to the app is just as easy.

Contents

Introduction

Congestion, don't we all hate it? The Dutch road system is heavily used, travellers need to take into account that travelling will usually take more time than could normally be expected. Traffic demand has increased yearly and will do so for the next years. Especially in the *Randstad* area, when an accident happens on a highway, travelling time in the entire road system will usually increase substantially. This leads to increased costs in extra fuel used and environmental concerns for the extra CO2-emission. On top of that, a lot of production time is lost since 68% of the congested traffic consists of commuters going to or coming from work [1].

Figure 1. Congestion leads to loss of time and money and imposes environmental issues

The Dutch government is trying to gain control of the situation by using several strategies, such as promoting *Smart Working*, carpooling, public transport and tax-friendly purchase of bikes.

The latest strategy is using innovative technology to fight traffic congestion. The Ministry of Traffic has reserved 25 million euro for the project "*Aanpak MultiModale reisinformatie*", with a main goal to "provide good travel information to every traveller in the year 2015, anytime, anyplace, so that the traveller can make an optimal and conscious choice, based on actual and reliable, nationwide, door-to-door travel information, both pre-trip and on-trip" [2]. This will result in a *Personal Intelligent Travel Assistant* (PITA), a mobile IT solution which will help the user in his/ her travelling pattern.

Fighting the congestion is of great social relevance, since it affects most of the Dutch inhabitants. This does make projects about Travel Assistants interesting to participate in. Social relevance of this project is obvious, almost everyone wishes for easier access of the highway system in the Netherlands, especially in the morning and late afternoon.

The government cooperates with The Netherlands Organization for Applied Scientific Research (TNO). TNO is a knowledge institute that converts science into applications. It provides services for governments, businesses and public organizations. The mission statement is: *"TNO connects people and knowledge to create innovations that boost the sustainable competitive strength of industry and well-being of society"* [3]. TNO is set up around seven research themes:

- Healthy Living
- Industrial Innovation
- Defense, Safety and Security
- Energy
- Built Environment
- Information Society
- Transport and Mobility.

Within the research theme Transport and Mobility, TNO is looking for ways to make traffic more reliable, safer, cleaner and quieter. In cooperation with government and industry, the research focuses on technological innovation as well as the influence of human behavior. The program Sensor City is a good example of successful use of technological innovation. Sensor City Assen is a project in which a large scale measuring network is constructed in the Dutch city of Assen [4]. Several hundred sensors were placed throughout the city. These sensors collect data about traffic loads, temperature, sound, or air quality. The use of sensor system technology creates new insights by combining all data from the sensors. The sensors act like the artificial sense organs of a model which adapts real-time to all information [5].

Figure 2. In the Sensor City project, IT is used to influence travelers

Within Sensor City, there are two subprojects, Sound and Mobility. The subproject Sound tries to measure and understand urban sound patterns. Sensors register sounds and identify it. With this sound map, applications are developed to reduce noise pollution. For example, microphones were installed in a nightlife area that trigger aggression camera's at a certain level of noise that indicates a fight [6].

The basic idea of the subproject Mobility is to make journey information available from and for the travellers in the country. It focuses on three goals: (1) generating reliable floating car data with measurements from the sensor grid, mobile devices and in-car systems; (2) travel time predictions based on that data and traffic models; (3) creating technology for future information services and nationwide travel paying systems. TNO has targeted the end of the year 2014 to achieve the goals for this subproject. Within the first goal, mobile devices are of great important, because the smart-phones of today are widespread amongst travelers and are packed with sensors and computing power.

With this personal device that the traveler has, combined with systems within the vehicle and in the infrastructure, influencing a journey can be achieved at three levels. This is visualized in figure 3.

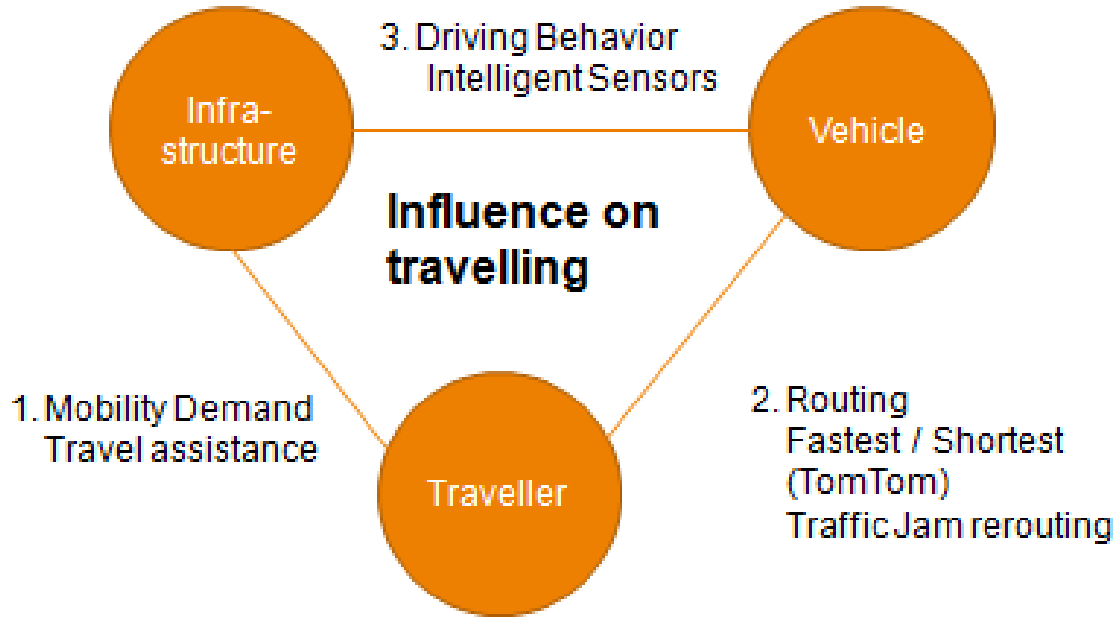

Figure 3. Three ways to influence a journey

First, the Mobility Demand. The personal device can inform the user with information from the infrastructure and can support the journey decision making process. For instance, when heavy congestion is noticed by the infrastructure, the advice could be to work at home for a couple of hours, use the public transport system or leave early to be able to arrive on time. This way, the demand to be mobile can be changed.

Second, on-trip routing. Navigation systems within the vehicle can help the traveler take the best (fastest or shortest) route to the destination. The traveler can even be rerouted to avoid a traffic jam.

Third, the driving behavior. Intelligent sensors within the infrastructure can interact with the vehicles driving by. For instance, sensors can regulate the speed of a vehicle or make the vehicle break in case of a detected accident further up the road. Also, traffic lights can be influenced when the sensors measure road load is too high.

1.1 Problem Definition

This project is part of the Sensor City Assen program within the theme "Mobility" of TNO and is set up under responsibility of the work group "Price and Information stimulation". The central question in this program is "how can travel behavior be influenced so that the individual and collective interest will be served?"

The problem we must face in this project is "How can an Android device be used to influence the departure time of a journey the user has to make?"

1. TNO wants a running prototype that advises the user of the best departure time for the next journey. The prototype must function on the Android platform.

2. By examining the calendar of the user, the app must know what journeys the user will have to make. However, the app should also be prepared to be able to work with another source for journeys, like a travel prediction algorithm.

3. The app must communicate with the Travel Time Predictor (TTP) from Model IT (Tripcast) to determine the most probable time to complete the journey. However, the app should also be prepared to be able to work with a TTP from another supplier.

4. Knowing the time to complete the journey, the app must calculate the best time to leave for that journey and advise the user about that with alerts.

5. The app must log all travel data in a database. This data must be synchronized at least daily with a TNO server. TNO will use this data to improve the quality of Service of TTP's.

6. The development must be documented properly, because of future expansion plans or integration of the functionalities within other projects.

A TTP can make a reliable prediction by combining traffic patterns with weather data and actual congestion information. Making a TTP is not part of this assignment. The Tripcast TTP, made by Model IT, is shown in figure 4. The difference between a static (Google) and dynamic (Tripcast) TTP is shown to the right of the figure. The dynamic TTP accounts for an extra 22 minutes travel time because of expected congestion.

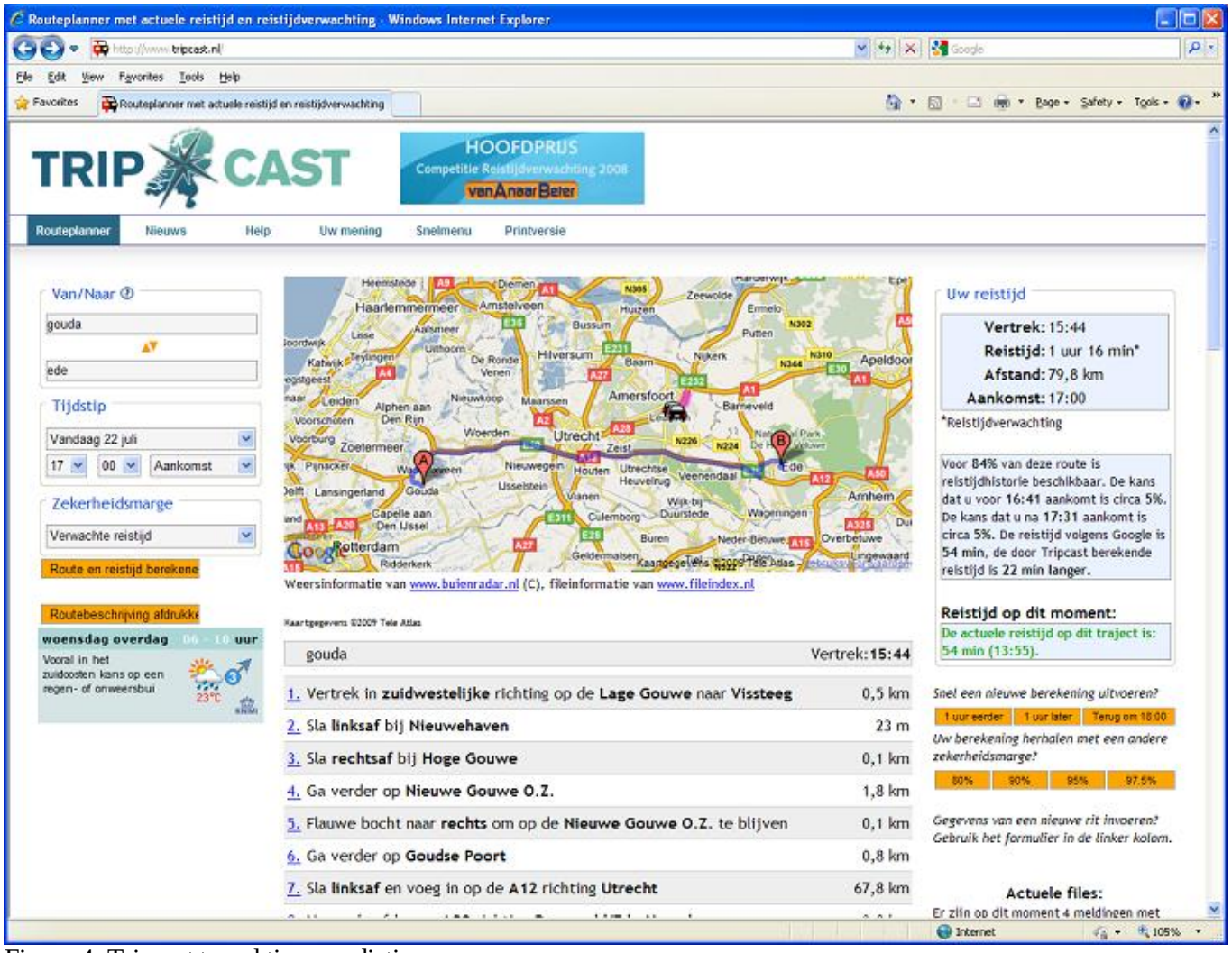

Figure 4. Tripcast travel time prediction

This assignment does imply several research challenges. Will a user give personal travel information in exchange for advice about the best departure time? In fighting congestion, it is difficult to predict the behavior of a traveller in terms of the conflicting interest between the individual and the collective. For instance, carpooling is in the interest of the collective, when a substantial part of travellers will do it, all will benefit with much shorter travel times. However, carpooling still is not common in the Netherlands.

Another research challenge will be to develop an application on the Android platform, which is still in development. Some features have not yet been standardized, other features lack documentation.

The final research challenge we face is the user interface of a mobile device. With the training at TU Delft concentrating on development for desktops, the project group will have to cope with the specific demands a mobile touch screen imposes on the developer. Several screen sizes and resolutions are available. The three inch HTC in figure 5 has a resolution of 240 by 320. The massive seven inch Galaxy Tab in figure 6 has a resolution of 1024x600.

Figure 5. HTC device running on Android

Figure 6. Samsung Galaxy Tab is the largest smart phone available

1.2 Used Methodology

The project will be executed in seven phases. In table 1 all phases are explained, together with the documents that were the result of that phase. Although this resembles a waterfall model, the phases are not strictly separated; a phase can start when the previous one is not finished yet. This will prohibit losing time in the first weeks of the project when a phase takes more time than planned. To monitor the progress, a formal meeting with the TU Delft assessor is planned weekly. A meeting with the TNO assessor is planned once every two weeks. For the BSc. coordinator, two-weekly progress reports and a timesheet are written.

Table 1. Defined phases of this project

Phase	Goal	Weeks	Documents
Plan	Plan the project to ensure that the results will be delivered on time	1	Plan of Approach
Research	Research the field to be able to create a state of the art solution	1,5	Orientation Report
Require-ments	Define the demands on the system to make sure that the right result is created	1,5	Requirements Document
Design	Design the best possible solution	1	Design Document
Imple-mentation	Coding to realize the solution	4	Code and Javadoc
Testing	Test the implementation for quality insurance	1	Test Plan and Test Results Document
Documen-ting	Write documentation to enable future development and prove capability of project group to assessors	1	User Manual, Developer Manual, Final Paper, Presentation, Trac data

Plan: a proper planning will help to deliver the product at the right time. In the plan of approach, expectations of the involved people are defined and a weekly planning with targets is listed. It contains a list of products to be delivered. It also contains a list of dependencies the project has on resources made available by TNO and TU Delft, like server access and work environment. The plan will have to be approved by the assessors of TNO and TU Delft. The format of the plan is based on Prince2, a method for project management. This phase was supposed to be executed in a week, however the document was approved in the fifth week.

Research: trying to create a state of the art application requires a lot of research, both in the field of Travel Time Prediction and in Mobile App making. On top of that, learning the finesses of the Android platform and selecting the right Android tools required extra examination. The research will also be published (in Dutch) separately as " Orientation Document", a mandatory step in bachelor projects at TU Delft.

Requirements / Design: When four people are to program on the same application simultaneously, agreeing on the requirements will help to create a solution that is shared by all people involved, project group, TNO and TU Delft. When the outline of the application is designed, it is critical to have the same view of the chosen, best possible solution. The design drawings are based on UML, the default modeling language. To describe the quality characteristics of the application in the requirements, the standard ISO9126 is used [7]. In the "Requirements Document", this is included.

Implementation / Testing: The most time-demanding phases will be partially integrated in this project, as an agile way of software engineering. Rules of software engineering will be held strictly:

- After each code update, the programmer will test the code. When no compile errors appear, the code will be released by a commit;
- Class names are clear and uniform;
- All issues will be tracked in a track wiki;
- Each class will have explaining comments in the code; this will be checked with a doc generator;
- There will be no deviations from the chosen design patterns;
- The Test plan describes the way testing will be executed; a significant amount of time will be reserved for the testing phase. The results will be summarized in a document called " Test Results".

Documenting: This will not be a stand-alone project within TNO. The documentation is needed to further develop the application or use its functionalities in other projects. The documentation will also be used to prove the competence of the project group to the thesis committee. The documents will also include a presentation and a demonstration movie.

Figure 7. Acceptance Testing is the final stage of the test phase

Related Work

2.1 User Interface Design

To influence users, an application must help a user in his/her activities and must be nice to use. This requires a simple and intuitive GUI, knowing that desktop GUI demands are different from mobile GUI demands.

The Golden Rules of Interface Design of Schneiderman [8] are a good starting point for interaction design. However, these rules were not specially designed for mobile devices. Gong and Tarasewich [9] have extended these Golden Rules so they can be used for mobile devices. The result is shown in table 2. Gong and Tarasewich also stated additional guidelines, which can be found in table 3. It is clear that our application will have to be service-oriented, low on user attention (functioning in the background) with little comprehension needed to function.

Table 2. Golden rules of Interface Design

Enable frequent users to use shortcuts	The desire to reduce the number of interactions and to increase the pace of interaction
Offer informative feedback	For every user action, there should be some system feedback, proportional to the action
Design dialog to yield closure	Organize actions into groups with a beginning, middle and end; give informative feedback at the completion to give the user a sense of accomplishment
Support internal locus of control	Give users the sense that they are in charge of the system. Design the system to make the user the initiator of actions
Strive for consistency	Interaction should be platform independent, the user might use an application on several devices
Reversal of actions	Allow a user to undo actions, this will encourage exploration of the application
Error prevention and simple error handling	Small buttons are easily pressed on wrongly so design the system to avoid serious errors
Reduce short-term memory load	In a mobile environment, a user has to deal with more distractions than on a desktop computer and users often perform other primary tasks besides the device, so trust on recognition of icons instead of memorization of commands

Table 3. Mobile Interface Design guidelines by Gong and Tarasewich

Design for multiple and dynamic context	With other activities and distractions around the user, allow operation with 1 hand or even 0
Design for small devices	Mobile platforms will continue to shrink, make buttons large enough or use speech input or output
Design for limited and split attention	Mobile users are frequently multitasking, so be prepared that the user will not have a full focus on the application
Design for speed and recovery	Users need to quickly save work and resume it later without loss
Design for top-down interaction	Present information through a multi-level mechanism and let the user decide to zoom into detail
Allow for personalization	Mobile devices are personal, users will set them to their own preference
Design for enjoyment	Although functionality and usability are key elements, users want an aesthetic user interface

2.2 Travel Assistance

Rothkrantz [10] describes a Personal Intelligent Travel Agent (PITA) as an application running on a handheld device, providing communication between a traveler and traffic information sources. It guides the user with an up-to-date personalized advice about the way of transportation, based on current and historic travel data. A PITA relieves the user of having to find the best route and travel opportunity to the destination.

Radu [11] has examined the user requirements for a PITA. Since this project will result in an application with PITA-like functionalities, the results of the research of Radu will be relevant for this project. In figure 8, the importance of methods of interaction is shown. The research states that interaction should not be handled by a traditional keypad.

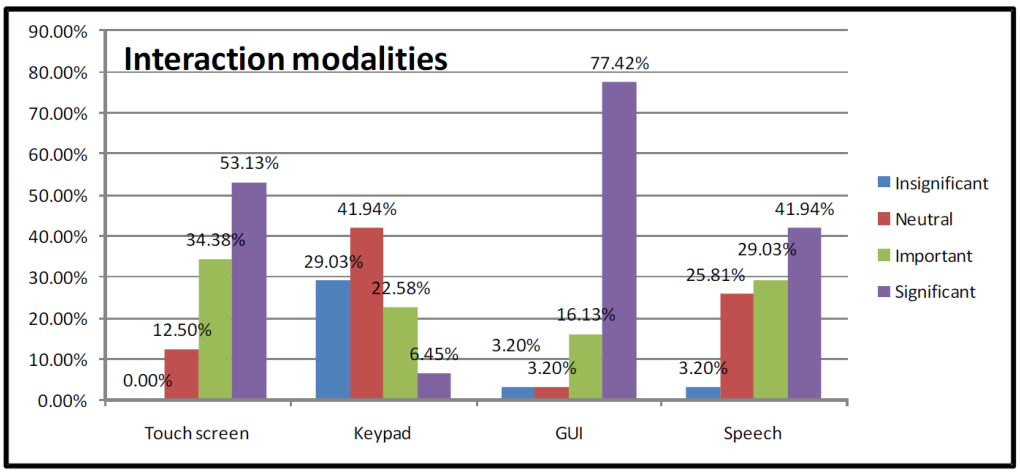

Figure 8. The importance of methods of interaction

The most important demand on a route is to have the shortest traveling time as seen in figure 9. It seems logical to optimize the journeys in our application for the shortest time. According to this research, most people want to have a suggestion for an alternative activity from the agenda when delay time came over an hour.

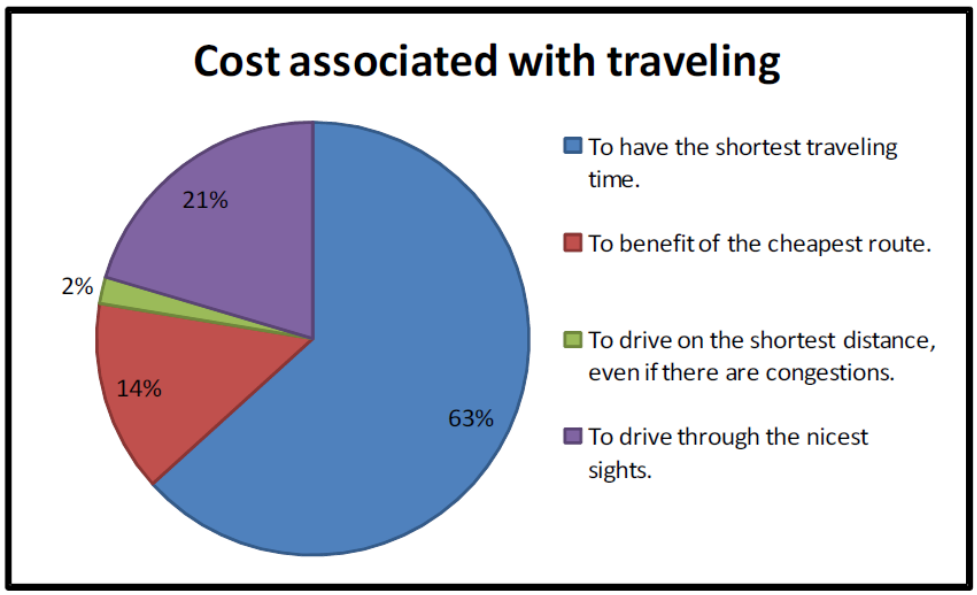

Figure 9. The most important demand on a route

In conclusion, Radu makes the following recommendations for PITA functionalities:

- Interact with the agenda of the user;
- Ease of use: The user just wants to travel from A to B and wants to be notified of any changes in the travel plan;
- Automatically update the software;
- A calculator for the total journey cost (time, money and environmental impact);
- Interact with a database of speed trap camera's;
- Interact with the car stereo for possible sound overlap;
- A possibility to contact alarm services.

Some of these advices will form the base of the application (for instance interacting with the agenda of the user), but due to time limits, some will be left out (for instance the automatic software update).

2.3 Travel Time Prediction

A very important aspect of Kate is to calculate the expected departure time for each appointment. For this, we needed an application that calculates the travel time between two places. If we would ignore all the factors that affect the traffic passage, this would not have been a very difficult task. However, not considering factors like traffic jams, accidents and the current traffic flow, would have us ending up with an idyllic travel duration. This duration would, probably, be shorter than the actual travel duration and we would not want that. What we wanted was a dynamic travel time predictor (TTP).

A lot of research has been done in designing dynamic TTP's. There are numerous methods one could use to achieve this goal. Radu [11] describes three approaches for this. The first approach is to use (mean) historical data in combination with current traffic and upstream traffic. Another one was to use time series analysis. Time series analysis is a popular way to predict travel times, because of its great potential for online implementation. The third approach she describes is by way of artificial intelligence. Neural Networks have been proven to be quite suitable for this job.

All these approaches or a combination of them could have been used to design a dynamic TTP, but given our time span, this would be quite a challenge. Fortunately, we did not have to build our own predictor. TNO provided us with a route planner that could do just what we needed. This route planner, Tripcast, was developed by Model It and has been awarded the title of best travel time predictor 2011 by the Ministry of Infrastructure and Environment.
An important issue to keep in mind with predicted travel times is reliability. It is common knowledge that travelers prefer the shorter travel duration. However, when it comes to reliability they prefer the longer but more reliable travel time over the shorter but less reliable one [12]. The way to ensure higher reliability depends on the way reliability is defined. If we would define reliability as "the probability that a particular part of road can be achieved within a certain travel time", then better reliability could be ensured by reducing the speed limits or by limiting the amount of traffic on the roads. These solutions would however probably not be accepted by society.
A possible solution might lie with the demand side in this. If we would introduce road pricing, travelers would be forced to reconsider their departure times. This is an interesting view because of the possibilities that lie within applications as Kate. For example by offering an alternative departure time which would result in a shorter travel time.

2.4 Similar Applications

When Problem Analysis and Design was complete, the project group found an app for the iPhone that had just been released. This American app, OnTime, offers comparable functionalities as Kate and will be released on the Android platform in the future [13]. The project group decided to pause the coding phase and research OnTime. After analyzing it, we concluded that it targeted a slightly different user. OnTime has more functionalities like a navigator system. It is half like Kate and half like TomTom. We concluded that OnTime posed no threat to our bachelor project. Some functionalities are presented identical, like the push-message alert system while the app runs in the background. OnTime differs from KATE in that it features functionalities that are not currently present in KATE such as multi-modality for driving and walking, changing the locations for an arbitrary appointment by looking up addresses and managing separate calendars can all be done in the application itself.
However it should be noted that OnTime does not have a reporting system and the TTP system cannot be adjusted. Kate on the other hand can generate weekly and monthly reports and can work with any TTP.

Figure 10. OnTime GUI

In figure 10, one can get an impression of how OnTime looks, from left to right are shown the Events, Event and Address Lookup menu. The Events menu gives an overview of all upcoming events. The Event menu contains all relevant information about an event, it also offers the possibility to indicate whether to use the event or not, to give directions and change the location of an event. In the Address Lookup menu the user searches for an address which gives a list of possible addresses and also sets that address to an event.

2.5 The Android Platform

TNO has specifically requested an app for the Android platform. This new platform is a software stack for mobile devices that includes an operating system, middleware an key applications. The Android SDK provides the tools and API's necessary to develop applications on the Android platform using the Java programming language. Android is still in development, the common release as of April 2011 is 2.3. In December 2011, the new 4.0 version will be released. Within a few years, Android has become the market leader in sales today. In figure 11, the market share of mobile operating systems as of April 2011 is shown. As illustrated in figure 12, it is expected that Android will account for almost half the sales in the year 2015.

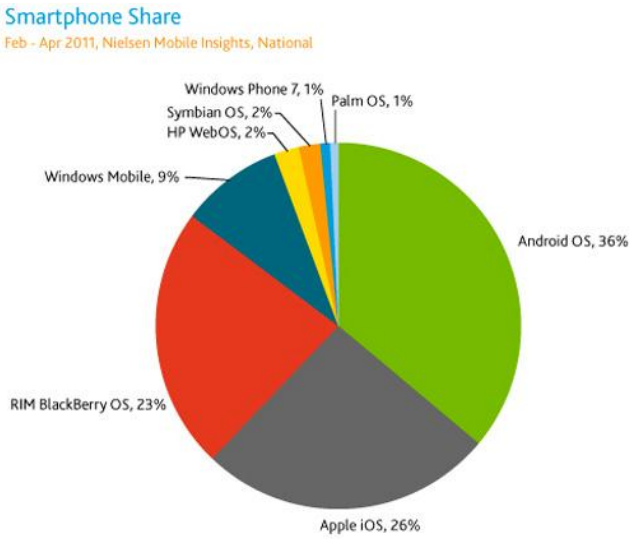

Figure 11. Market share in Smartphone sales, February – April 2011, www.Nielsen.com

Worldwide Smartphone Operating System 2011 and 2015 Market Share and 2011-2015 CAGR (listed alphabetically)

Operating System	2011 Market Share	2015 Market Share	2011-2015 CAGR
Android	39.5%	45.4%	23.8%
BlackBerry	14.9%	13.7%	17.1%
iOS	15.7%	15.3%	18.8%
Symbian	20.9%	0.2%	-65.0%
Windows Phone 7/Windows Mobile	5.5%	20.9%	67.1%
Others	3.5%	4.6%	28.0%
Total	100.0%	100.0%	19.6%

Figure 12. Expected market share in Smartphone sales in 2015, www. Nielsen.com

The main features of Android are [14] [15] [16]:

- Application Framework enabling reuse and replacement of components;
- Dalvik Virtual Machine, optimized for mobile devices;
- Integrated browser based on the open source WebKit engine;
- Optimized graphics powered by a custom 2D graphics library, 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional);
- SQLite for data storage;
- Media support for common audio, video and still image formats (MP3, AMR, Mpeg4, JPG, PNG, GIF);
- Default use of camera, GPS, compass, accelerometer, Bluetooth, GSM and WIFI on Android device (hardware dependent).

Supported by the Open Source world, Android has quickly become the mobile platform primary choice since its launch in 2008 [17]. Main research companies predict that Android will have a 45% smart phone market share in the year 2015 [18].

2.6 Conclusion

The research has given new insights to the project group about mobile application development and the science of traffic behavior.
The development of the user interface will be done with the golden rules in mind. Actions will be icon based and intuitive, the app will not rely on the memory of the user and it will work in the background, ready to react on the user's demand.
The research has shown that making a TTP on our own was not feasible, so it was ruled out. The Tripcast TTP has won many prizes for being the best available TTP. We will use it as an information service. It is clear that the majority of people normally would want the shortest route possible for their journey. However, they favor a longer but more time-reliable route over a faster but uncertain one.
On Android, there are no similar applications. There is a comparable application available for Apple iPhone. This App, OnTime, tries to integrate PITA functionalities with a navigator like TomTom.

Tools

Selecting the right tools is extremely important in this software development project, because the short amount of time available forces the project group to develop the product as fast as possible. The tools can either cost a lot of time to learn, or help to gain a lot of time automating standard project jobs, like testing and generating Java statistics. The project group has done research to select the right tools for this project and decided to use the following tools.

3.1 Software Development

Using Java was added as an extra requirement by TNO, because it is the default programming language at the organization. That posed no problem, because the Android platform is entirely based on Java. Java programming could be done with just a Notepad and a command line compiler. However, an Integrated Development Environment (IDE) will support the software engineering process substantially, for instance with refactoring, auto-completion, quick navigation, automatic access to helpdocs and the possibility to work on the code as a team, so Notepad was quickly ruled out.
Choosing an IDE for Java is difficult, because of the large amount of choices: BlueJ, JDeveloper, NetBeans, JBuilder, Greenfoot and Eclipse are all used in our work field.
However, nowadays Eclipse is the leading IDE [19] and has become the *de facto* standard for non-desktop Java development [20] [21]. Eclipse is the only IDE that can directly invoke the Android development tools within the IDE [22]. The project group has unanimously worked with Eclipse for the last three years, so there will be no learning curve, a vital aspect in this short project. Therefore, the open source packet **Eclipse** will be used as IDE for Kate.

Developing for the Android platform requires the **Android SDK**. It provides tools and libraries that are necessary to develop applications on Android devices [23]: Eclipse integration, Debugger, Error handler, Syntax checker and Emulator. The Android SDK is available on Windows, Mac OS X (Apple) and Linux. However, the project group only has access to Windows computers, so Android SDK for Windows 7 will be used. Figure 13 shows the settings tab of our application running in the emulator.

Figure 13. Emulator in Android SDK, showing the settings of Kate

Working with Eclipse and Android SDK makes it possible to use the powerful **Android ADT**. These *Android Development Tools* are a set of plugins for Eclipse that turn the IDE into a fully integrated Android development workbench. The tools help to test and debug code, design the specific Android GUI, use localization, manage resources (with XML) and export .apk files to distribute the app [24].

Delivering maintainable and scalable source code is a vital requirement for TNO. As part of the developers manual, reference documents for the source code must be present. Javadoc is the standard tool for this task. However, the group has chosen the tool **Doxygen**. Like Javadoc, Doxygen is a tool for writing reference documentation from within the code and can cross reference documentation and code [25]. Doxygen can generate better abstracts than Javadoc, necessary in this project to prove to TNO that the code is maintainable [26]. It is also easier to configure and can generate more kind of graphs and PDF's [27].

3.2 Test Environment

A good programmer will always test the fresh source code. Getting rid of bugs at the beginning of the development process is the most time-efficient method. Testing should therefore be executed as close to the development as possible. Testing atomic snippets of code is called Unit Testing. Unit Testing in Java is almost always done with JUnit. The Android SDK has a built-in JUnit functionality. However, running tests in the Android emulator is very slow. Building, deploying and launching for one unit test will often take more than a minute. This is not suitable is you're performing dozens of test daily.

Figure 14. Robolectric is a unit test framework designed for Android

A much faster way to perform unit testing in an Android environment is to use **Robolectric**. Robolectric is a unit test framework that runs directly in the desktop Java Virtual Machine and handles most tests within seconds [28]. It is uniquely designed for the Android platform and handles almost identical as JUnit so the learning curve is minimal.

A part of functional testing and integration testing is code coverage. Code coverage is a measurement of the part of the code that is executed while the test was run. This way, it will be clear which snippets of code will not have been tested. **EclEmma** is the open source code coverage extension for Eclipse. The tools inspects the code directly and is very fast [29]. EclEmma will give an insight into the quality of the code and will help to prepare the project group for code inspection by SIG.

3.3 Revision Control

Revision Control is the management of changes in the code. This way, a team of developers is able to work on the program at the same time. The revision control system tracks and controls changes to the source code.

The client-server development approach in this project restricted the number of choices for a revision control system. Systems like GIT and SVK are primarily focused on distributed software development. Proprietary client-server systems like Microsoft TFS were not an option, financially. The Dutch initiative CVS, developed in 1986 and very popular in the past century, is outdated. Apache Subversion (abbreviated **SVN**) is the most commonly used open-source revision control system [30]. SVN offers user-friendly code-sharing and revisioning. The project group already has some experience working with SVN, so there is virtually no learning curve at the start of the project. Figure 15 shows the distribution of all commits in the implementation phase.

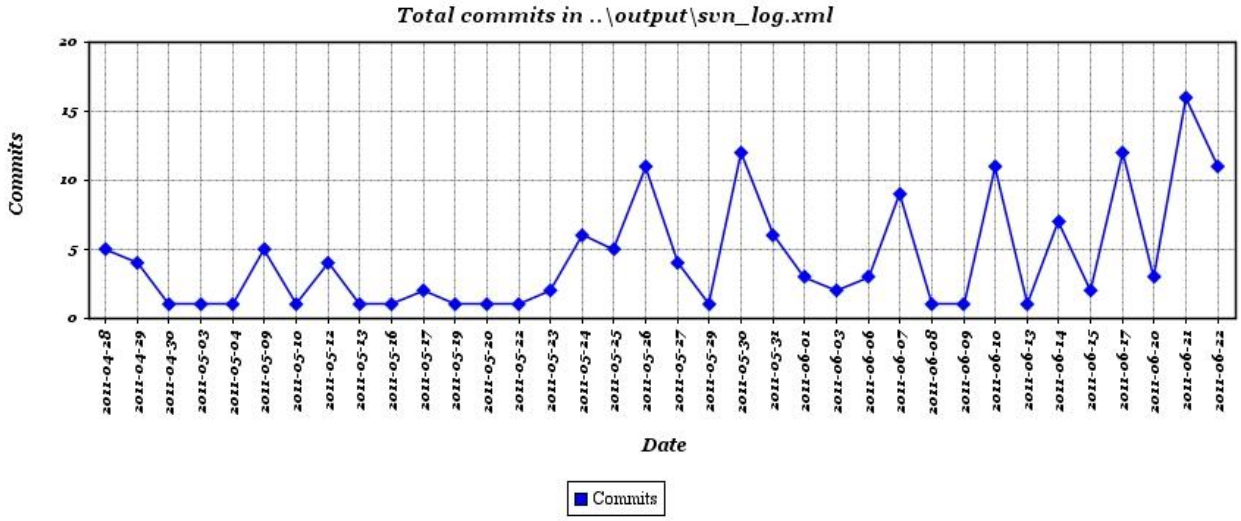

Figure 15. Total commits on Kate within SVN

SVNstat generates summaries of SVN usage[31]. Automatically generated graphs and charts show commit activities and other useful information. This will also help to improve planning capabilities of the project group members. Figure 16 shows the total lines of code of Kate. The dip in the end is the reaction to the comments from SIG. SIG suggested some alterations on the code, resulting in the deletion of some obsolete code. This is explained in more detail in the chapter Implementation.

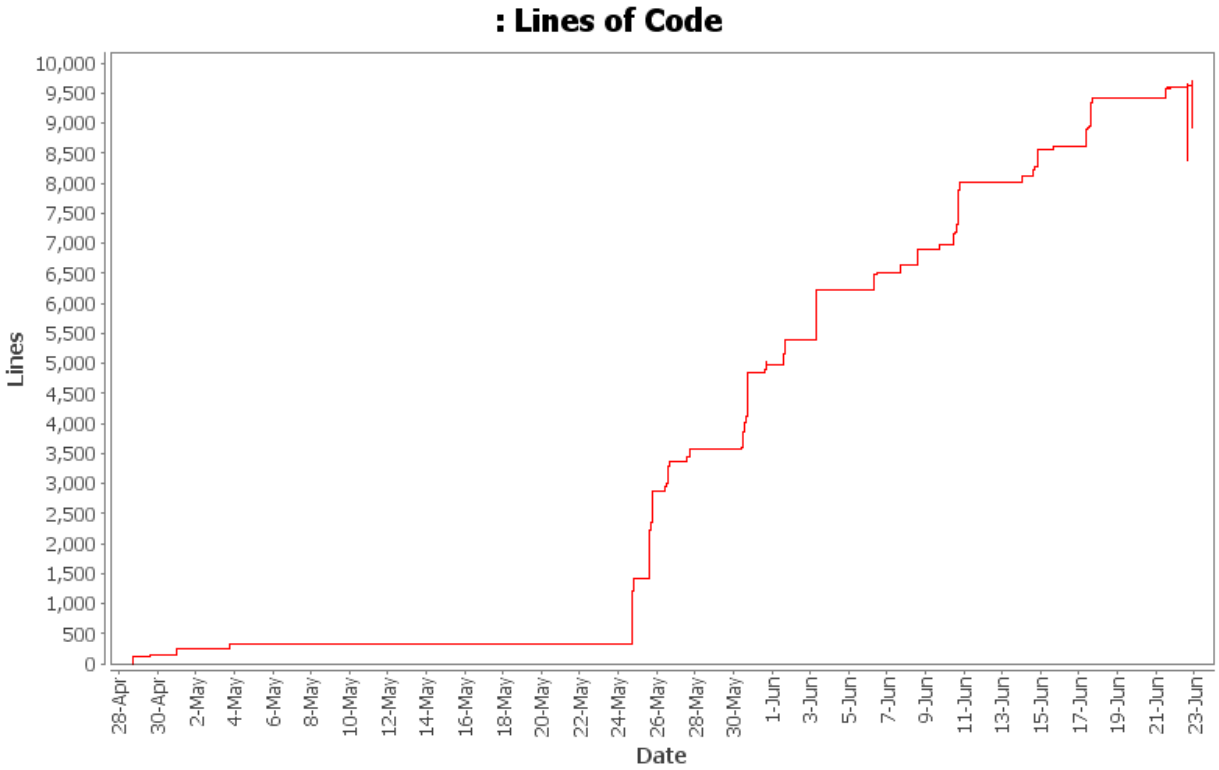

Figure 16. Total lines of code of Kate, generated by SVNstat

With Eclipse and SVN as tools for this project, it is possible to use **Subversive**, an Eclipse plug-in that integrates SVN into Eclipse. With Subversive, SVN can be used within the Eclipse environment so that Eclipse project control and task management is integrated with version control [32].

As four programmers with a different amount of coding experience are working on the source code simultaneously, it is only natural that bugs will appear. Also, a programmer can discover a issues in the code of another programmer that should be corrected. To keep track of these issues and bugs, an issue-tracking system is used. The logical choice is **Trac**, an open source, web-based issue tracking system designed to interface with SVN [33]. Trac offers simple to use issue tracking and a Wiki environment for documentation. This is a necessary feature, as TNO has planned to further develop the app with another project group to add other functionalities. Trac will offer insight into development of the app and will reduce the learning curve for the future development team.

3.4 Internal Android Communication

During the design phase of the project it was debated over what would be the best method to acquire information regarding the calendar of a user. The two options that came up were to make use of the internal **Android Calendar** or an external one such as Google Calendar. Our initial research into these two choices had revealed that using an external calendar was the most viable option.

Integrating an external calendar with KATE could be realized by using the **GData** protocol that provides many services such as Google Maps and Google Calendar [34]. Within this protocol we would limit ourselves to the **Google Calendar Data** protocol which provided sufficient documentation for implementing this service into the application [35].
The key reason behind this first choice is that internal Android Calendar does not have a public API and isn't part of the Android SDK, thus making it difficult to integrate it with Kate [36]. A common issue is the lack of forward compatibility which is due to the changing of the content providers in Android version 2.2 or higher [36]. The content providers are essential for reading events stored in the internal Android Calendar. Considering our inexperience with programming for the Android platform this seemed as the easiest way.

However, during the early stages of implementation it was discovered that the Google Calendar Data protocol does not work well with the Android platform. We were put in quite a predicament seeing as some time has been put on the Google Calendar Data protocol. After doing some further research into the matter the decision was made to use the internal Android Calendar instead, since it provided many more benefits over the Google Calendar Data protocol.

Some of the benefits are that Kate does not require an internet connection to read the calendar, thereby reducing the consumption of battery life. Support for Google Calendar also exists within the Android Calendar. Android users can sync their devices with a Google account and in the process sync their calendars as well. Another benefit is that users would not be required to give their username and password to Kate to gain access to Google Calendar, making the support for it rather redundant.
The issue of forward compatibility will still linger until Google decides to make the Calendar API public. For the time being we have attempted to fix the issue by letting Kate check which version the Android device is running to determine which content providers to use. This solution allows Kate to use the Android Calendar for Android version 3.0 or lower.

3.5 Data Storage

An important goal of Kate is to log certain information of the user to a central database, which in Kate's case was based on a TNO server. For reasons of efficiency and battery life we chose to build two databases. One would be internal and the other on the server. Instead of having to send a request to the server every time this logging information becomes available, we just save up data on the internal database for a little while and send it all at once. Another great advantage of this strategy is that we don't lose any data if the server goes offline or the internet connection disappears. In these cases Kate would just collect all the logging information and store them locally until the server becomes reachable again.

For external data storage we did not yet know which database engine TNO would work with, so we chose to use MySql. However, we wanted to make it rather easy for TNO to switch to a database engine of her choosing. For this reason we created an abstract layer for the Java code to approach. This was achieved by using a technique that is called Object-relational mapping (ORM), creating an abstract layer for the Java code to approach. Since, in a Java environment, Hibernate is the default ORM software on desktops and because Hibernate supports a wide range of database engines, we used the Hibernate framework.

For internal data storage, the Android SDK has built-in support for **SQLite**. The SQLite engine is ideal for mobile phones, since it uses only 180 Kb of memory and demands little processor capacity [37]. Other engines like MySQL, Oracle or PostgreSQL are much heavier on resources and have no Android support as of May 2011 [38]. The SQLite query syntax differs slightly from the MySQL syntax, with which the project group has worked before. For the simple operations, only needed for this project, the project group expects no problems in compatibility.

There is always a type mismatch between a traditional relational database like SQLite and an object oriented programming language like Java. As mentioned before, Hibernate is the default ORM software on desktops in a Java environment. However, it has no default Android support (as of May 2011) so it is difficult to configure on an Android system, it requires a lot of tweaking and it is very demanding on resources, making it awkward to use on Android [39]. **ORMLite** is a lightweight framework, designed to work on mobile devices (it has built in Android support) with limited resources [40]. It is easy to learn and will help to reduce the time needed to realize the data storage code.

While a relational database is used to store internal data, **XML** is used for other forms of data interchange, like GUI components, string resources (for easy localization) and communicating with the Tripcast TTP. It is not certain that all TTP's will use XML as means of communication with clients, but it is likely that most will offer such functionality, because the easy to learn XML format is the default data format on the Internet [41]. The user manifest (see chapter Implementation) on Android must also be written in XML.

Design

"One can get an idea of the concept behind this app by the following illustration. Imagine having to travel from place to place for appointments you have with clients and you are currently in a meeting. But you would like to know when the appropriate time is to end your current meeting and leave for the next one. These days you're left with no choice but to figure this out for yourself and this could be time better spent on something else. This is where this new application comes in.
It is designed to read an agenda or calendar of the user in order to extract appointments and other useful information such as their current location. Based on user preferences and travel information, the application will advise a user when to leave in order to arrive on-time for an appointment. In essence our application acts as a personal secretary." (Brainstorm project group, April 2011)

We did not use a formal software design method for this phase. Instead, we formulated our own plan to design this app, since we all had a certain image of it in our head. The plan consisted of four steps:

- A brainstorm to make these images in our head clear for the other members; this led to several drawings of the app and certain parts or screens;
- A requirements definition meeting, in which we assembled all possible requirements;
- A MoSCoW analysis. MoSCoW helps to make clear what requirements the group agrees on that are necessary or not for the app;
- Drawing the UML graphs Class Diagram, Use Cases and Sequence Diagram.

After the first brainstorm, it was clear that the app we were creating would act like a Personal Intelligent Travelling Agent. As stated in the chapter Research, a PITA is an advanced travel information system which has potential to induce a shift towards a more efficient use of the existing transportation systems. Known PITA's aim at offering services that combine traffic and public transport information, static and dynamic information, pre-trip en on-trip information and personalize this information [42].
The focus for our app is on two main subjects: to provide accurate departure times for the Android user and to provide journey logging information to improve the service from the Travel Time Predictor (TTP) at TNO.

The app can be divided into seven main parts:

1. The Main Activity;
2. The Appointment Manager;
3. The Trip Manager;
4. The Alert Manager;
5. The Logging Manager;
6. The Report Manager;
7. The Database Manager.

The relations between these parts are visualized in figure 17.

The Main Activity is the starting point of the app. It starts up all the processes and monitors the outcome. It builds up the GUI and sends it to the OS for further handling. It activates the Appointment Manager.

The Appointment manager listens to the calendar of the user. When a new appointment is detected, the Appointment manager will try to resolve the arrival information for that appointment. If not successful, the user will be asked details about the appointment. If successful, the appointment data will be sent to the Trip Manager.

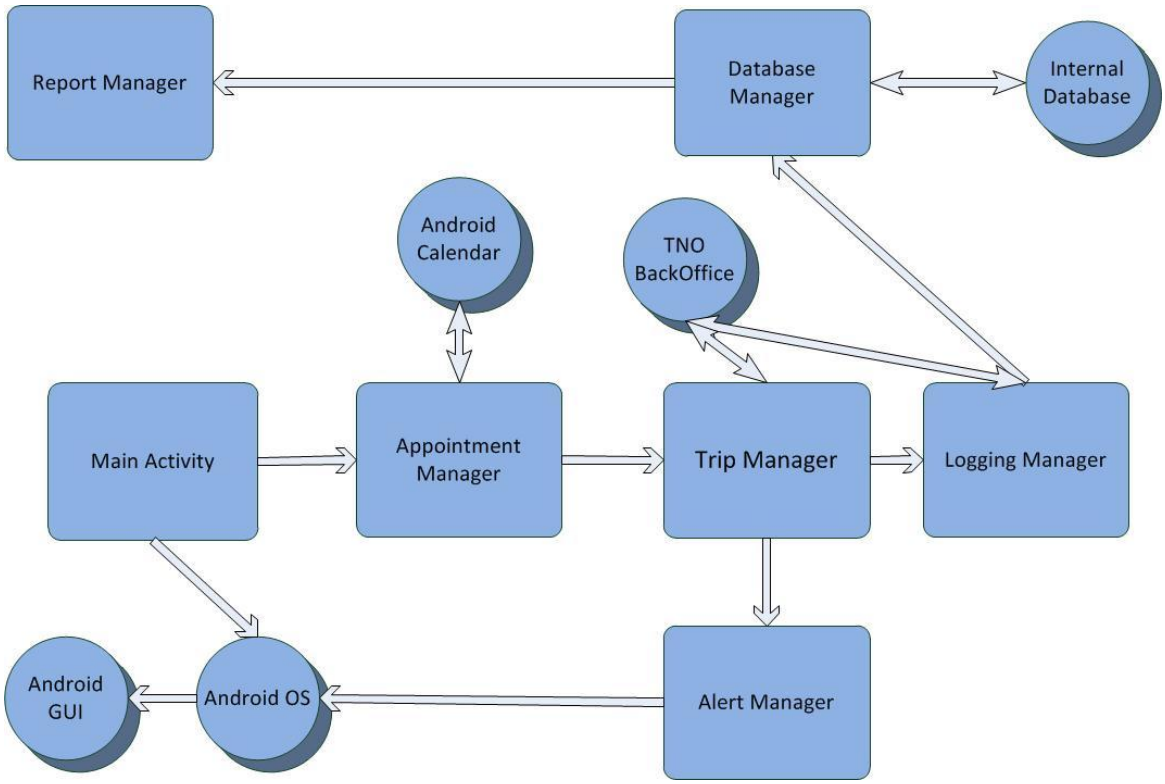

Figure 17. Overview of Kate

The Trip manager is the central point of intelligence of the app. It controls the entire process (except the reporting). When the Trip manager receives appointment data, it will communicate with the TTP in the TNO BackOffice to determine travel time and will then calculate best departure time. This is sent to the logging manager for logging purposes. The appointment data is then communicated with the Alert Manager to create alerts. The interface with the TTP from Tripcast is a XML-file request / service. Communication is done by sending a standard XML file to the server as a request. The Tripcast TTP interprets the XML for a departure location, destination, arrival time and some parameters that are set default in our case (like the parameter "use an automobile?" is always set to on). Tripcast sends back a XML file containing all necessary data. Kate filters out the "95% reliable travel time". That is the time with which 95% of the journeys on that route can be made in time.

The Alert manager will calculate the right alert times based on the user settings and trigger the alerts before the departure time is imminent. Alerts are handled like Android notifications and put on the notification stack via the Android OS.

The Logging manager logs the trip data into the internal database. Once a day it tries to connect with the log server in the TNO BackOffice. If successful, it will flush the data to the BackOffice. If not successful, it will try again the next day.

The Report Manager makes an abstract of the data in the internal database and generates weekly and monthly reports about the journeys. In Kate the user can get a weekly or monthly report of how much hours have been spent on travelling from appointment to appointment, the distance traveled in kilometers and the amount of times the user has arrived late for an appointment. This information can be beneficial to a user that might be concerned about fuel consumption, wishes to reduce tardiness or travel time.

The Database Manager handles all communication with the internal database. An internal database is used for collecting necessary journey data. The usage of in internal database besides the log database at the TNO BackOffice will reduce employment of capacity from the battery and cpu.

A possible user story is that of the business man with lots of appointments today:
"Our businessman awakens and grabs his phone. Both he and his secretary has made appointments for today, five in total. He presses the Kate icon on his phone. After displaying the TNO logo for three seconds, Kate will resolve the appointments from the calendar. It calculates the expected travel times for each appointment, making sure that the journeys are executable. The businessman arrives at the office in Delft, checking his email and talking to his colleagues. Then, his Android phone starts to vibrate. He know this is the first alert from Kate, to let him know he has to leave in 15 minutes for his meeting in Rotterdam. The first alert is earlier than he expected, so he knows that the journey will take more time than usual. He continues the conversation with a colleague about a problem in a project. Then there is a vibration from the phone in his pocket again. The second alert from Kate indicates that he should make himself ready to leave. He finishes the conversation and starts packing for the meeting.
Later on, during that meeting, the vibration starts again. Kate alerts him about the next appointment in Amsterdam. Due to accidents, Kate has calculated that the journey will take 35 minutes more than usual, so thanks to the vibration the businessman will now work this meeting to a finish to be able to arrive in time for the next appointment. Before, when Kate did not exist, he would depart at a regular time and conclude, with loads of anger and frustration, on the road that he would never reach the next appointment in time. "

A complete overview of all the classes included in the application can be found in the class diagram that is delivered on a CD as a Visio document, along with the use cases and the sequence diagrams. Parts are shown here, but the entire diagram is too large to print. The initial class diagram has been designed prior to the implementation of the system and contained a total of 21 classes. In the mean time, it has developed and expanded to include almost 70 classes.
The class diagrams of two of the more important main parts of the system are shown in figures 18 and 19.

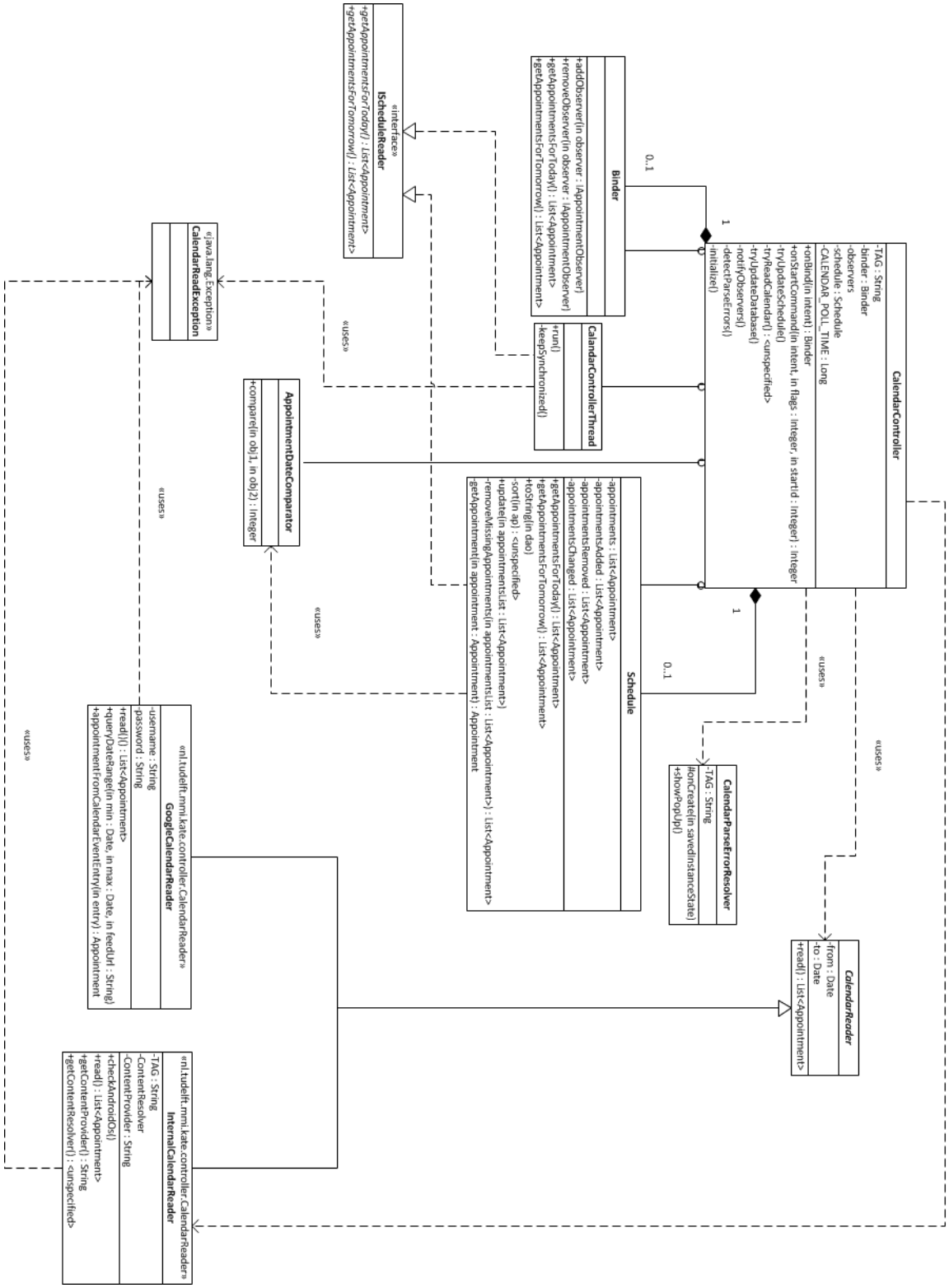

Figure 18. Part of the class diagram

Figure 18 shows the links between the classes that deal with the calendar. The main class in this subsystem is the CalendarController. This class is responsible for ensuring that everything that has to do with reading the appointments from the calendar is dealt with properly. The CalendarController has a couple of inner classes that support it in its tasks. These inner classes are:

- AppointmentDataComparator
  This class is a custom comparator for Appointmenta. Used for sorting Appointments by date in ascending order.
- Binder:
  The Binder class provides a simple functionality of synchronous method invocation.
- CalendarControllerThread:
  This is a thread class for CalendarController so that it can run separately from other services. This class will ensure that everything to do with the calendar will be synchronized.
- Schedule:
  This class will contain and manage the appointments read from the calendar.

The CalendarReader classes deal with the actual reading of the calendar. CalendarReader is a superclass that has two subclasses. The first one, GoogleCalendarReader, reads the online Google calendar that is connected to the Google account of the user. InternalCalendarReader reads directly from the calendar on the Android phone. When reading from a calendar, an exception might occur for example due to inaccessibility to the calendar. These exceptions have been classified as CalendarReadExceptions. Another problem that might occur when reading appointments from a calendar is the problem of incorrectly entered appointments or incomplete appointments (appointments missing a place etc.). CalendarController detects these errors and passes them to CalendarParseErrorResolver to appropriately deal with them.

Figure 19 shows the class diagram of the back office package. The back office handles incoming requests from the application. These requests are either requests to store the log data in the main database or to calculate the travel time for an appointment. The CommunicationManager listens for such requests. When it detects a request, an instance of the CommunicationServer class is asked to read this request and write back a reply. The CommunicationServer determines the type of the request and sends the request to the appropriate RequestHandler. Depending on the request type, the DataStorageRequestHandler either stores the data in the database or the TravelTimeRequestHandler calculates the travel time. This is done by using an instance of TravelTimePredictionReader. In the current state, that will always be TripcastReader. TripcastReader sends an HttpRequest for the travel time to the Tripcast predictor and reads the result.

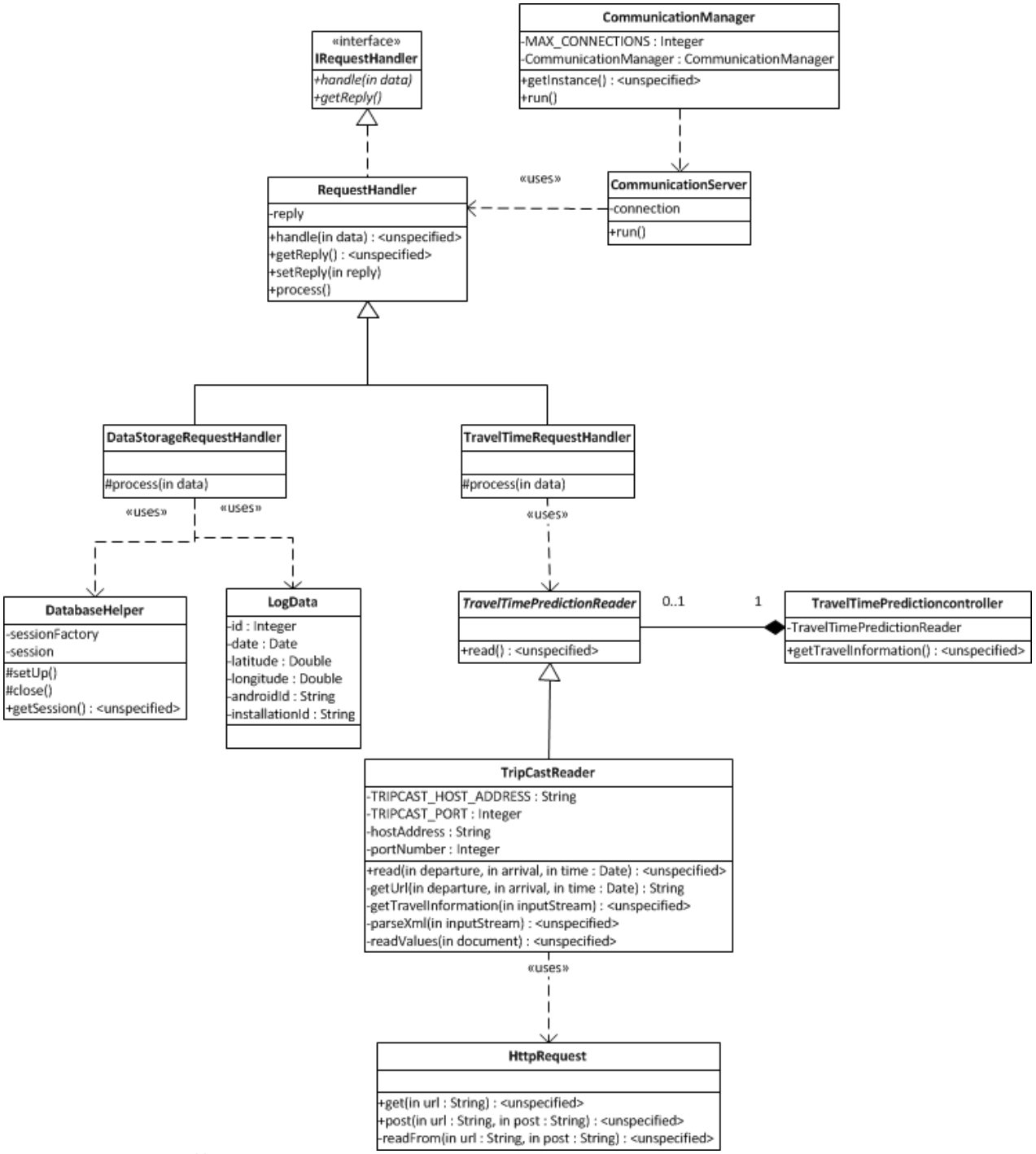

Figure 19. Back office class diagram

While designing the user interface, Android gives the developer many possibilities to present data on the screen. Vocal control was ruled out, because the alerts had to be discrete. For instance, when you are in a meeting, the vibration alert for the next appointment will be preferred over a voice saying that you have to leave.

```
[main.xml]

<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@android:id/tabhost" android:layout_width="fill_parent"
android:layout_height="fill_parent">
<LinearLayout android:orientation="vertical"
android:layout_width="fill_parent" android:layout_height="fill_parent"
android:padding="5dp">
<TabWidget android:id="@android:id/tabs"
android:layout_width="fill_parent" android:layout_height="wrap_content" />
<FrameLayout android:id="@android:id/tabcontent"
android:layout_width="fill_parent" android:layout_height="fill_parent"
android:padding="5dp" />
</LinearLayout>
</TabHost>
```

Figure 20. XML interface declaration

The main screen uses a tab interface, which makes sub screens accessible via tab buttons. The XML code in figure 20 does not declare anything in regards to interface contents, it just tells Android that it will use tabs in this interface. The code that controls the main screen adds tabs by linking sub activities to the Tab Host and configuring their tab buttons, see figure 21. Use of a tab interface gives the user great control over what part of the application he or she would like to see, and having the interface specified in this way gives great control and flexibility to the programmer to create and add more tabs. Figure 22 shows an example of a tab interface of a popular social app with four tabs. On the right is the Tab interface of Kate, normally with four tabs, but with the debug option on, now with five tabs. The tab shown in the figure displays all the upcoming appointments that occur today or tomorrow.

```
[HomeActivity.java] (abbreviated version)

public void specifyTabInterface()
{
// specify the tabs
Resources res = getResources();
TabHost tabHost = getTabHost();
TabHost.TabSpec settingsTabSpec =
tabHost
.newTabSpec("settings")
.setIndicator("", res.getDrawable(R.drawable.ic_tab_settings))
.setContent(settingsIntent);

// add the specifications to the tab host
tabHost.addTab(settingsTabSpec);
}
```

Figure 21. XML tab definition

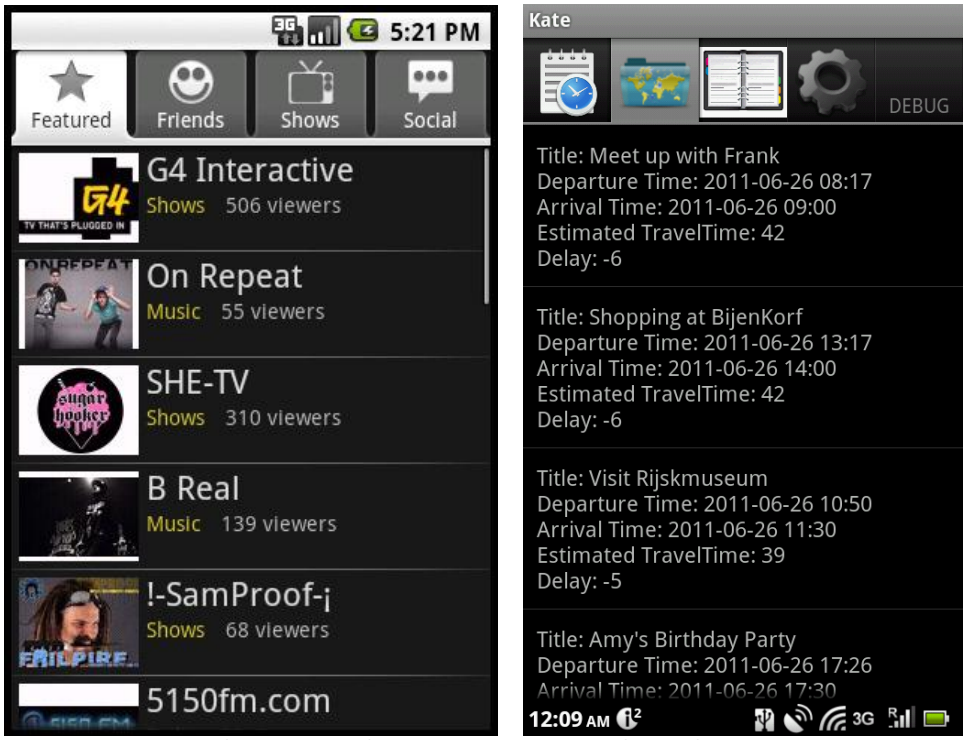

Figure 22. Example of a Tab interface and the Tab interface of Kate

After trying out the possibilities that Android gives for the user interface, the Tab layout was chosen for Kate. This is the most user friendly layout that Android offers. Many famous apps use the same layout, like Phonebook, Whatsapp or Movie finder. The main advantages are:

- Very simple navigation through the functionalities;
- Information can be found by clicking only one button, all menus are only one level deep;
- Actions are represented by symbols instead of letters. It is based on recognition, not remembering the functionalities;
- It is scalable; adding or deleting a tab is very easy, making it easy to add a tab for debugging (or a new functionality made by TNO), for example.

We tested the tab interface of Kate with the project members as selected users. After that, two people that were not project members, got the app and tried to work with it. The conclusion was that the learning curve was very small, mainly because of the easy layout of functionalities.
Within minutes all the involved were able to work with the app and get alerts for journeys they planned.

Implementation

5.1 Structuring

The implementation phase of a project has often proved to be the most intense and time consuming phase. With this wisdom in mind, we attempted to reduce the stress factor in this stage by making a strict planning. We started out by planning two sprints of both two weeks in which we would work towards a couple of milestones we had set.
When planning this phase we divided the system up to three main categories, which are the graphical user interface, the back office and calendar handling. Having four persons in this team, this seemed optimal. This way we could assign end responsibility for each of these categories to a team member and have one team member left to bear the responsibility of 'gluing' all components together and for the integration.

5.2 Process

Before we started the programming phase, we first worked on the design of the system. We started off by creating use cases, class diagrams and sequence diagrams which we extracted from our MoSCoW model. This way we all had a clear idea of the way we wanted to structure the implementation. We used these designs to start the implementation by implementing a 'skeleton' that adhered to the design specifications. So in the first few days we had already implemented all of the classes of the UML diagrams, however without most of the functionality.
Having comprehensively defined the MoSCoW made it easier for us to set daily and weekly milestones to work towards. After setting up the skeleton for the code, we started adding functionality on the basis of the MoSCoW. When we started adding functionality, we realized that, even though we had planned everything carefully and thoughtfully, we would have a hard time sticking to the original planning. Four weeks of programming seemed to be insufficient, so we added an extra week to the first sprint. This summed up to a total of five weeks of full time programming. The increase in time was needed because we didn't take into account that the learning curve of developing on an Android platform would take so long. Also, the documentation on Android is not optimal yet, so we had to figure out many things ourselves.
During implementation we kept updating the design specifications, because many things had to be implemented differently because of the Android interface or because the functionality ended up more complex than we initially thought it would be. Eventually we ended up having implemented all of the must-haves and most of the should-haves we defined in the MoSCoW model.

5.3 Coding

Implementing an application on the Android platform required more than just the understanding of Java and the Android API. We had to code specific parts of the system in XML. One of these parts was the preferences settings. Figure 23 shows a part of the file preferences.xml in which some of the settings and the poll rate are defined.

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
        <PreferenceCategory android:title="@string/preference_category_user"
                android:visible="false">
                <EditTextPreference android:key="username"
                        android:visible="false" android:title="@string/preference_username"
                        android:summary="@string/preference_username_summary" />
                <EditTextPreference android:key="password"
                        android:visible="false" android:title="@string/preference_password"
                        android:summary="@string/preference_password_summary"
                        android:password="true" />
        </PreferenceCategory>
        <PreferenceCategory android:title="@string/preference_category_system">
                <CheckBoxPreference android:key="alerts_enabled"
                        android:title="@string/preference_alerts_enabled"
android:summary="@string/preference_poll_rate_summary"
                        android:defaultValue="100" />
                <PreferenceScreen android:numeric="integer"
                        android:title="@string/preference_time_to_alert_screen_summary"
                        android:summary="@string/preference_time_to_alert_screen_summary">
                        <ListPreference android:id="@+id/preference_time_to_alert_1"
                                android:key="time_to_alert_1"
```

Figure 23. Part of Preferences.XML

One of the defining features that Kate would have is the ability to give travel information based on the user's calendar. In order to perform this task it becomes paramount to have a component in Kate that interacts with the calendar of an Android device and processes it into something useful for Kate.

At first it seemed straightforward to simply read the events calendar from the device, but as mentioned in paragraph 3.4 Android does not have a public Calendar API. The risk of using a non-public API is that, should Google decide to change the Calendar API for later versions of Android, then Kate would no longer be able to read the Android Calendar. In the past the calendar content providers have been changed. Figure 24 shows the method to tackle this issue by detecting which version of Android it is running to determine what content providers to use for reading the internal calendar. It is possible that other changes have been made in the content provider, but we only resolved one of the changes since the internal calendar is only being used to read events.

```
if (Build.VERSION.RELEASE.contains("2.2"))
{
                    // Android 2.2 or higher.
                    contentProvider = "com.android.calendar";
              }
              else
              {
                    // Android 2.1 or lower
                    contentProvider = "calendar";
              }
}
```

Figure 24. Part of CheckAndroidOS method

In the calendar component the AppointmentManagerService class is tasked with the duty to manage all interactions with the Android Calendar. It asks the InternalCalendarReader to extract the events from the Android Calendar and convert them into appointments which are then placed in a schedule. These appointments are also stored in Kate's internal database. The schedule consists of appointments that will take place today and tomorrow. The AppointmentManagerService is run as a thread to ensure that it can act independently from other activities or services and periodically requests that the schedule and database be updated to real time changes in the internal calendar. With the observable pattern used in our implementation any changes made in the schedule will immediately be notified to all observers, such as the user interface and the TripManagerService.

Figure 25 shows how the InternalCalendarReader reads the calendar events from an Android device. Some of the limitations to using this code are that there is no clear way to retrieve a unique event ID, making it difficult to uniquely identify appointments. Recurring events all have the same event ID which makes changing certain event properties such as location easier, but detecting whether an event has changed almost impossible.

```
List<Appointment> appointmentList = new ArrayList<Appointment>();
            // Read events from selected calendars
            Uri.Builder builder;
            String[] projection = new String[] { "event_id", "title",
"eventLocation", "begin", "end", };
            String sortingOrder = "startDay ASC, startMinute ASC";

            for (String id : calendarIDs)
            {
                  // Specify the time span
                  builder = Uri.parse("content://" + contentProvider +
"/instances/when").buildUpon();
                  ContentUris.appendId(builder, beginRange.getTime());
                  ContentUris.appendId(builder, endRange.getTime());

                  // Query the calendar for events.
                  String selection = "Calendars._id=" + id;
                  Cursor eventCursor = contentResolver.query(builder.build(),
projection, selection, null, sortingOrder);

                  // For a full list of available columns see
                  // http://tinyurl.com/yfbg76w
                  while (eventCursor.moveToNext())
                  {

      appointmentList.add(createAppointmentFromEventCursor(eventCursor, id));
                  }
            }

            return appointmentList;
```

Figure 25. Part of the InternalCalendarReader

5.4 Manifest

The Android Manifest is an extra step in implementing software on the Android platform. The manifest presents essential information about the application to the Android system. Android must process this information before it can run the application.

The contents of the Manifest is:

- Package name for the application;
- Components of the application (activities, services, broadcast receivers, content providers). These declarations let Android know what the components are and under what conditions they can be launched;
- A declaration of the permissions the app must have in order to access protected parts of the API and interact with other apps, including the minimum level of the Android API that the app requires;
- A declaration of the libraries that the app must be linked to.

Any permissions necessary will be asked to the user on installment.

5.5 Pseudocode

Although it is difficult to explain briefly and clearly how the program is implemented, it usually is useful for outsiders to have some knowledge about this process. For this reason we designed pseudo code which roughly describes the structure and functioning of the system. As the pseudo code is written for better human understanding, details of the program which don't contribute to that better understanding are omitted.

```
do initialize
while (program is running)
        if (phone calendar adjusted)
                update appointments in Kate
        end
        calculate travel information
        if (time to alert)
                show notification
        end
        log [longitude, latitude, id, date]
        save travel information to local database
        save log data to local database
        if (connection to server available)
                flush log data from local database to server database
        end
end
```

5.6 Feedback from SIG

After our first sprint we had the opportunity to send our code to the Software Improvement Group (SIG) and only a couple of days later, we received feedback on it. The feedback was primarily focused on maintainability. Our code scored almost five stars on their maintainability model, which means that the code has above average maintainability. The reason why the perfect score of five stars was not achieved, was the lower score on Interfacing Unit and Unit Size.
For Unit Interfacing, SIG looked at the percentage of code units with an above average number of parameters. Usually this indicates a lack of abstraction and it leads to confusion in calling the method and in most cases to longer and more complex methods. However, SIG did not recommend to change these classes, because they were classes that were adjusted from the Android Platform Source Code. Adjustments to these type of codes would make it harder for bug fixing and was therefore not recommended. Instead, we were advised to properly document the what changes have taken place on which version of the Android Platform Source Code, so that this may make it easier to remove the code later.
For Unit Size, SIG looked at the percentage of code that was above average long. In our system, we had some methods that were too long. For these methods, such as 'read' in the InternalCalendarReader class, we were advised to look for separate pieces of functionality to find out what we could refactor to separate methods.

Their final words of wisdom regarded the (unit-)tests. We had already made a good start on automation testing, but it wasn't yet a permanent part of our developing process. SIG recommended to integrate this to ensure that future adjustments to the current functionality is working.
In these last couple of days, we tried to implement all the feedback into our project. The adjusted Android Platform Source Code has been removed and it's functionalities have been replaced. The long methods have been cut into shorter pieces and hash-codes have been used in classes as Appointment.java. We integrated the unit tests and with this completed the automation tests.

With all the comments from SIG taken care of, plus the positive feedback that accompanied those comments, we expect a maximum rating of five stars for the second review.

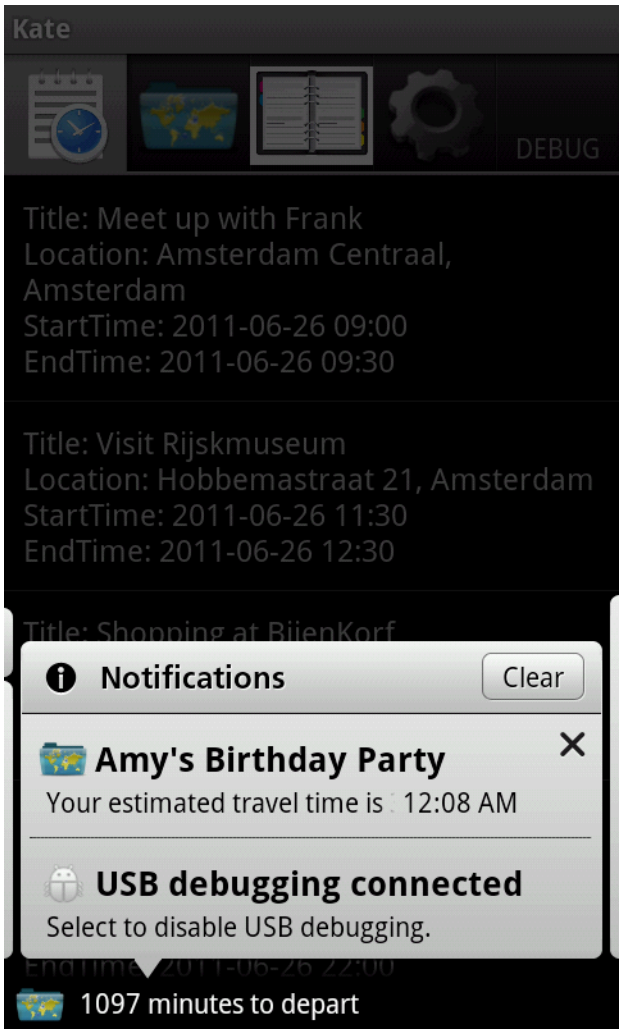

Figure 26. Kate Notification

Testing

If designers and programmers wouldn't make mistakes, then testing really would not be necessary. But, since they are human, they will make mistakes [43].

6.1 Test Planning

The project group executed the coding and testing in a combined phase. This agile approach ensured that, when code was written, it was tested to compile. Only code that could be compiled would be committed. After the system tests, which took most of the time in this phase, several other tests were administered before the acceptance test finally led to accepting the application by TNO. Table 4 shows the planned test activities. It is a part of the test plan. The test strategy is explained in this plan. The starting point of this strategy was to test as soon as possible to be able to detect errors fast. Testing was divided into five test parts.

Table 4. Planning of Test Activities

Testtype	Datum	Tijd	Opmerkingen
System tests	23-05 t/m 17-06	92 h	
- Unit test	23-05 t/m 12-06	38 h	Robolectric/ Junit
- Functional test	03-06 t/m 10-06	27 h	Built-in in Eclipse
- Integration test	06-06 t/m 12-06	27 h	Android SDK
Usability test	10-06 t/m 15-06	6 h	Android Emulator and Android Devices
Performance test	13-06 t/m 17-06	8 h	Android Devices
- load test / volume test	13-06 t/m 17-06	4 h	Test on speed / capacity
- Android platform	13-06 t/m 17-06	4 h	Test execution on at least two Android platforms with different processor speed
Authorization test	15-06 t/m 17-06	4 h	internal (Android OS, SQLite, ORMlite and GData) and external (TNO BackOffice)
Acceptance test	17-06 t/m 26-06	8 h	product owner

In most software development projects, testing will take up 10% of the available time.
In this project, 7% of the available time was allocated for testing. This deviation comes from the fact that the project group are students, the members are still learning and some project time was allocated for competence development of members. Because this is a university project, more time is allocated for documentation of the project, as more documents have to be completed.

6.2 Test Results

To ensure that we build a proper application, we performed a couple of tests on it. These tests can be divided in four major categories, which are system tests, usability tests, performance test, authorization test and acceptance test. In the remainder of this chapter we will discuss the results of these test.

6.2.1 System test

During implementation we made sure to test the individual objects by means of JUnit testing. This resulted in some unexpected results on some tests, which alarmed us to adjust the code. Eventually we made unit tests for every class except the classes in the back office. Those classes depend on a connection with the database and with Tripcast which made them hard to test separately with unit tests.
We implemented all the requirements that we discussed in the requirements document. Some of the requirements were however, eventually implemented and/or designed differently than we at first thought we would. These changes have primarily been a result of the lack of understanding we first had of the Android Platform.
The system was integrated continuously, so the integration testing also was somewhat done constantly. This resulted in the fact that we were quick to find errors that wouldn't pop up from the unit tests.
To see what is happening within the system when running an integration test, we implemented a Debug tab in Kate. All output on what the system is doing would be shown there.

In figure 27 Kate has a new tab build for debugging purposes. This functionality is primarily aimed at developers that want to know which background operations are being performed and if they have succeeded. It also provides the developers the flexibility to test if the GPS logging services is working to expectations while not being confined to the PC. In the regular app, this tab would be deleted (which is very easy in the tab layout of Android). However, this is a proof of concept and TNO requested this tab, so it was left in the final version.

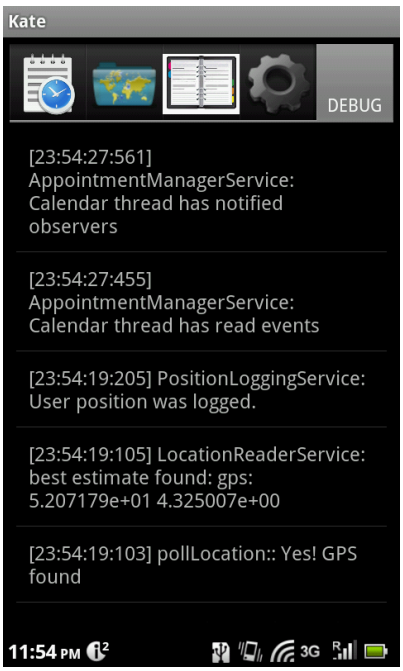

Figure 27. Kate Debug window

Usability test

We put the usability to the test by asking persons who were not involved in the design of Kate to test the usability and by testing it ourselves. The content and navigation and interaction of the application seemed to be fine. The recovery test was fine as well. Kate could handle wrong input, lost connections with the server or GPS and sudden shut down.

Performance test

We tested Kate on the Eclipse emulator, two different Android phones and an Android Tab. Kate worked on all of these devices in a similar manner. One of the disadvantages of Kate seemed to be the battery life. When implementing the system, we implemented all methods as efficient as possible to minimize battery usage. However, since TNO required Kate to send log data to the back office every minute and every time the user's GPS location would differ ten meters from its previous location, this requires quite a lot battery usage. When running Kate the battery life tends to decrease threefold.

Authorization test

The greatest part of the authorization tests was already covered by the integration test. We tested the use of all external sources and Kate had the correct setup and right to work with them.

Acceptation test

The acceptation test was run several times in the last couple of days. The first time we let an outsider use Kate and Kate seemed to work fine, but the interface wasn't optimal yet. Since we were trying to influence the user's behavior by Kate, this was a big issue. Kate had to be user friendly. We adjusted the interface a couple of times to the wishing of several test subject until they were all satisfied with the ease and use of Kate. It is now up to TNO to prove that an app can influence the behavior of the user, to benefit both that user and the collective as a whole.

Conclusion

The central question in this project was: "How can an Android device be used to influence the departure time of a journey the user has to make?"
Kate makes it possible to influence the departure time, with the alerts trying to make the user leave at the right time. Field test will eventually show if and how real people will let their journey times be affected by a mobile app. The requirements in the problem definition have been realized.

1. *TNO wants a running prototype that advises the user of the best departure time for the next journey. The prototype must function on the Android platform.*

Kate is a running Android prototype. The advices Kate gives have been tested, they take delay times into account, suggesting a departure time which will ensure just in time arrival. The application has been delivered to TNO as apk-package and as source code.

2. *By examining the calendar of the user, the app must know what journeys the user will have to make. However, the app should also be prepared to be able to work with another source for journeys, like a travel prediction algorithm.*

The app communicates with the Android Calendar. This Calendar can be synchronized to a Google Calendar very easily. It resolves any trips found in the agenda, stating a arrival place and arrival time. Departure place is resolved as the place of the last meeting, or the current location. The Appointment manager, part of the code, has been built modularly, the class designed for the Android Calendar can easily be switched with another class.

3. *The app must communicate with the Travel Time Predictor (TTP) from Model IT (Tripcast) to determine the most probable time to complete the journey. However, the app should also be prepared to be able to work with a TTP from another supplier.*

The app can send a trip request to the Tripcast TTP via the server. It receives a XML file containing all necessary data to determine journey time. The Trip manager, part of the code, has been built modularly, the class designed specifically for Tripcast van easily be switched with another class.

4. *Knowing the time to complete the journey, the app must calculate the best time to leave for that journey and advise the user about that with alerts.*

Journey times and last possible departure time have been determined by the Trip manager after communicating with the TTP. Based on the user settings about the times between alerts and the last possible departure time, alerts are pushed on the Android alert stack. The user receives those alerts with a vibration from the phone.

5. *The app must log all travel data in a database. This data must be synchronized at least daily with a TNO server. TNO will use this data to improve the quality of Service of TTP's.*

The app logs data like longitude and latitude of the user at a certain time. Also, trip requests are stored at the server. With this useful data ,TNO will better understand individual travel patterns and delays and can better predict future travel times on particular journeys

6. *The development must be documented properly, because of future expansion plans or integration of the functionalities within other projects.*

Along with the application, the following documentation is delivered to TNO:
Requirements Document, Design Document (with UML), Testplan, Test Results, User Manual, Developer Manual including Doxygen Javadoc and Trac data.

Although the project group has lost more than two weeks waiting for resources and learning the finesses of the Android platform, the application has been delivered on time and was accepted by TNO, so it is safe to say that the project has been a success.

A comparable app with PITA (Personal Intelligent Travel Agent) functionalities already exists on the Apple platform, but Kate is available on Android before the Apple OnTime app. Besides that, OnTime is more of a navigator than a travel agent.

Several ideas are stated for future development of Kate to gain commercial value for the app, like integrating other means of travel besides automobiles, or availability on other platforms than Android.

Android is a promising platform that is set up to support the Java developer by supplying tons of useful libraries and making it easy to use the sensors within the device.

7.1 Future Work

When implementing Kate version 1.0, we have had a lot of ideas for version 2.0. In decreasing order of importance, we advise the following extensions to the app:

- Travel prediction; based on the historic travel scheme of the user, predict which journeys the user will probably make today and suggest them to the user. This will make the app more independent from usage of the Google Agenda;
- Integration of other means of transport; the app makes the assumption that the user will be travelling by car. Integrate the usage of public transport, bicycle or carpooling;
- Manual input of a journey; when the user wants to demonstrate Kate, or just wants to know at what time he should depart without making an appointment in the Android Calendar, he should be able to manually insert a trip in the Kate queue;
- Integrate the app with navigator systems like TomTom; send the information about the arrival place and arrival time to the navigator app and start it *just in time* ;
- Platform Independence; the app is only available for the Android platform. With two out of three people using another platform in April 2011, usage of the app will grow faster when the app is available on other platforms like Apple iOs and Windows Mobile; however, this will not be easy, because the device handling is different and the platforms use different development languages (Apple uses Objective C, Windows uses .NET) while translation tools between the three platforms are not yet common;
- Using XML as default export; the app has XML support, necessary for communication with TTP's, but the logging uses direct SQL database communication. With XML export, the generated data could be used more general;
- More reports; the app makes default weekly and monthly reports for the user. More research can be done in the need for specific reports.

7.2 Discussion

In this project, we were facing a difficult software engineering challenge within very strict time limits. There were several difficulties in this project, like the unavailability of a work environment, both at TNO and at TU Delft. This resulted in extra responsibilities, but also in a kind of freedom that made the project very pleasant to do.
The lack of documentation of some of the essential Android code has made an extra challenge. The biggest problem however, was the opposite: over-documentation because of the many systems to implement something. For example, communication between Activity (interface) and Service (background process) can be done by a Binder, a Messenger, or using AIDL script. Within each of these solutions, there are several possibilities to implement the system. Apparently this is inherent to working on a platform that is still under construction.

The project group has sufficient Java knowledge, but implementing Java on the Android platform required an extra twitch during coding.
It is clear that Android is a promising, but not quite ready, platform that will definitely be the dominating mobile platform before the spring of 2012.

7.3 Recommendation

The project group has learned valuable lessons while making Kate. We would like to make the following recommendations for future bachelor project groups :

- Take into account that learning to develop for the Android platform will take approximately two weeks extra to get an understanding of the Android dialect. Compared to normal Java programming, programming Java on Android is comparable with a person from *Holland* trying to speak the Limburg dialect: technically it is identical, it looks identical, but it feels as something completely different.
- Use specialized development tools, designed for the Android platform. Using Android ADT, Robolectric and EclEmma has given us a substantial time advantage, while the learning curve for these tools was surprisingly short.
- GPS emulation in the Android SDK is not working properly. Use a real device to test GPS functionality in an app.
- Develop for Android 2.0 and higher. Lower versions are used by only 4.4% (and decreasing) of Android users [44] and require a lot of extra work because some libraries are not backwards compatible.
- Make clear agreements with the product owner about the boundaries of the project and secure this in a fixed document within the first two weeks.
- Decide on the best methodology to use and discuss it with the project mentor.
- Planning is crucial in such a short project. Make a planning in week 1 and take into account that implementation will take longer, plan to cope with a delay time of a week at the end of the project. Planning will be changed during the project, be prepared.

References

[1] Rijkswaterstaat, "Profiel van de spitsrijder. Wie rijdt er in de spits?", Rijkswaterstaat Adviesdienst Verkeer & Vervoer, 2006

[2] website Rijksoverheid, visited at May 20, 2011, http://www.rijksoverheid.nl/documenten-en-publicaties/kamerstukken/2009/07/02/ 20092606-aanpak-multimodale-reisinformatie.html

[3] website TNO, visited at May 25, 2011, http://www.tno.nl/content.cfm?context=overtno&content=overtno&item_id=30

[4] website Blik op Nieuws, visited at June 7, 2011, http://www.blikopnieuws.nl/bericht/109411/Miljoenensubsidie_voor_sensorstad_Assen.html

[5] website TNO, visited at May 5, 2011, http://www.tno.nl/content.cfm?context=overtno&content=nieuwsbericht&laag1=37&laag2=69&item_id=2010-03-16%2009:48:55.0&Taal=1

[6] website Sensor City, visited at May 5, 2011, http://sensorcity.nl/pagina/47/content/3

[7] E. van Veenendaal, *The testing practitioner*, Tutein Nolthenius uitgeverij, 2004

[8] B. Schneiderman, "Designing the User Interface - Strategies for Effective Human-Computer Interaction", Addison-Wesley, 1998

[9] J. Gong and P. Tarasewich, "Guidelines for handheld mobile device interface design", College of Computer and Information Science, Northeastern University, Boston, 2009

[10] L. Rothkrantz, Dragos Datcu and Martijn Beelen, "Personal Intelligent Travel Assistant, a Distributed Approach", TU Delft, 2005

[11] A.A. Radu, "Personal Advanced Traveller Assistant", TU Delft, 2010

[12] H. van Lint, H.J. van Zuilen and H. Tu, "Altijd zondag?", Transumo, 2005

[13] website OnTime, visited at May 12, 2011, http://www.ontimemobileapp.com/index.html

[14] website Android developers, visited at June 19, 2011, http://developer.android.com/guide/basics/what-is-android.html

[15] website Webkit, visited at June 19, 2011, http://planet.webkit.org/

[16] website Packet Video, visited at June 19, 2011, http://www.packetvideo.com/products/android/index.html

[17] website Marketing Charts, visited at June 19, 2011, http://www.marketingcharts.com/direct/android-claims-1-smartphone-platform-position-15695/

[18] website IDC, visited at June 19, 2011, http://www.idc.com/getdoc.jsp?containerId=prUS22871611

[19] website SourceForge, visited at June 13, 2011, http://eclipse-cs.sourceforge.net/

[20] D. Abbott, "Embedded Linux Development using Eclipse", Newness, 2008

[21] website InfoWorld. visited at June 13, 2011, http://www.infoworld.com/d/developer-world/sdk-shoot-out-android-vs-iphone-074

[22] website Android, visited at June 13, 2011, http://developer.android.com/guide/developing/index.html

[23] website Android, visited at June 13, 2011, http://developer.android.com/sdk/index.html

[24] website Android, visited at June 13, 2011, http://developer.android.com/sdk/eclipse-adt.html

[25] website Wikipedia, visited at June 13, 2011, http://en.wikipedia.org/wiki/Doxygen

[26] website Doxygen, visited at June 13, 2011, http://www.stack.nl/~dimitri/doxygen/manual.html

[27] website Stack Overflow, visited at June 13, 2011, http://stackoverflow.com/questions/225447/doxygen-vs-javadoc

[28] website Robolectric, visited at June 13, 2011, http://pivotal.github.com/robolectric/

[29] website EclEmma, visited at June 13, 2011, http://www.eclemma.org/index.html

[30] website Subversion, visited at April 25, 2011, http://subversion.apache.org/

[31] website SVNStat, visited at April 25, 2011, http://svnstat.sourceforge.net/

[32] website Subversive, visited at April 25, 2011, http://www.eclipse.org/subversive/

[33] website Trac, visited at April 25, 2011, http://trac.edgewall.org/

[34] website Google Data Protocol, visited at April 25, 2011, http://code.google.com/apis/gdata/docs/developers-guide.html

[35] website Google Calendar Data API , visited at April 25, 2011, http://code.google.com/apis/calendar/data/2.0/developers_guide.html

[36] website Jim Blacker, Accessing the internal calendar database inside Google Android applications, visited at April 25, 2011, http://jimblackler.net/blog/?p=151

[37] website SQLite, visited at June 13, 2011, http://www.sqlite.org/about.html

[38] website Android, visited at June 13, 2011, http://developer.android.com/guide/topics/data/data-storage.html

[39] website TouchTech, visited at June 13, 2011,
http://www.touchtech.co/blog/ormlite-for-android/

[40] website ORMLite, visited at June 13, 2011,
http://ormlite.com/sqlite_java_android_orm.shtml

[41] website IBM, visited at June 13, 2011,
http://www.ibm.com/developerworks/opensource/library/x-android/

[42] Prof. Dr. H.J.P. Timmermans, NWO research programme "Gedragsaspecten van PITA", 2007

[43] M. Pol, "Testen volgens TMap", 2nd edition, Tutein Nolthenius Uitgeverij, 1999

[44] website Platform Versions, visited at June 13, 2011,
http://developer.android.com/resources/dashboard/platform-versions.html

Definitions and Abbreviations

ADT	Android Development Tools
AIDL	Android Interface Definition Language
AMR	Adaptive Multi Rate, a variant on MP3 for mobile devices
App	Application for a mobile device
CVS	Concurrent Versions System, a management system for source code that keeps track of changes made to the code and allows for easy recovery of older versions
GPS	Global Positioning System, a system to determine the longitude and latitude of a device
GUI	Graphical User Interface, the interface through which the user interacts with the program, the graphical view that is manipulated by the touch screen and buttons on the mobile device
IDE	Integrated Development Environment
MoSCoW	A way of assigning priorities to project requirements, according to the following rules: Must-haves must be implemented in this project Should-haves should be implemented if time allows it Could-haves would be nice to have, now or in the future Won't-haves will not be done this version and are left for the next group of developers
ODBC	Open Database Connectivity, a standard programming interface for accessing a database
On-trip	The time during the journey
ORM	Object-relational mapping, converting data between a relational database and an object-oriented language
PITA	Personal Intelligent Travel Assistant
PNG	Portable Network Graphics, a mobile image format

Pre-trip	The time before the time to depart
SIG	Software Improvement Group, a department of the TU Delft that checks source code from students for quality and efficiency
Smart Working	The flexible use of labor time and production environment instead of a fixed nine-to-five pattern at the office
SQL	Structured Query Language, a language for communicating with databases and altering database structure
TNO	Netherlands Applied Scientific Research Organization
TTP	Travel Time Predictor, an automated service that predicts the time needed to travel between a departure and an arrival location, given a departure or arrival time.
XML	Extensible Markup Language, a markup language used to describe data

Appendices

A. Process Evaluation
B. List of Figures
C. List of Tables
D. Assignment Description (Dutch)
E. Plan of Approach (Dutch)
F. Orientation Report (Dutch)
G. Requirements document (Dutch)
H. Design Specification (including UML) (Dutch)
I. Testplan (Dutch)
J. User Manual (Dutch)

Appendix A, B and C are required parts of this document by standards of the faculty and are included in this document.

The other appendices are the documents that were created during the project. They are delivered on a CD as a supplement to this document. They are only available in Dutch.

Appendix A. Process Evaluation

Group reflection

Looking back at eleven weeks of cooperation, we are proud of the final result. Maybe, if we had taken a week longer, more fancy functionalities could have been added, but we realized that would not add to the core of Kate, so we sticked to the original plan.
Time has flown by so fast. At first there was some doubt about the group composition, with diversity in study length, experience and background. Trying to make an application with a software engineer, a researcher, an information analyst and a manager required extra time to learn to work together.

As part of our initial planning, we would adopt an agile approach which would take elements from Scrum m(a developmental method or framework) as the main method of group consult. Our tight plan included weekly coding sprints in which we would set tight deadlines, hold a short daily meeting and end the week with a working and fully integrated prototype. Soon it became clear that we could not stick to it. While we had good ideals and planning, delays in getting the right resources and the lack of working space resulted in the loss of several production days. It took several weekends to get back on track. When a team member was not available for some days because of personal problems and another team member became ill at the end of the project, Kate started to get on our nerves. That resulted in some disagreements between members, but we managed to resolve most of the different points of view. Another important issue may have been resource availability. We were not assigned a stable work place and had to work with variable working places around the campus. While software is available on these computers, for Android programming it is preferable to have system admin rights. For instance, in order to install or debug our app on a physical device, appropriate drivers must be installed. The alternative to this is to use emulators, they are however limited in the functionality they offer compared to the actual device. Calendars, for example, are not available in the emulator, which meant that we could not properly test our main functionality using the campus computers. We were able to do testing by having one team member bring his own computers and delegate any required test work to this member.

At first, a weekly meeting with our professor seemed too much. However, we soon learned that the weekly meeting was a great result booster and helped to make the deadlines. We underestimated the amount of work that the final report required. The iterations between the professor and us improved the report vastly and the feedback was returned very fast, but the process consumed loads of time. With the experience we now have, we would definitely plan this project more strictly, with an extra week at the end for delivering the final report.

Individual reflection

-Loubna

Working on Kate has been a delight. I've learned more from this than I thought when we first started. On this program I was assigned responsibility on the functionalities in the Back office (server). Not having as much experience with programming as the average computer science student, this made me a little nervous about having the responsibility about this part. However, having some great programmers on the team who are willing and patient enough to help and steer when needed made the programming experience one of the best so far. I have learned a lot from this experience, especially how to build a maintainable system, since Jozua was always pushing to keep that in mind.

Since we had set a strict deadline for ourselves, we had made a strict planning from day one. This planning did not last long though. We updated the planning frequently, because we would have encountered unexpected problems, or we just did not plan enough time for a part. We eventually did manage to finish the project before the deadline, but it has cost Kate some functionality. We did not implement all the should haves and could haves, when we were actually planning to do so.

I think this group was rather strong in resolving issues that arose when members would have disagreements and problems that occurred with or between group members. We always solved any issues that occurred to the satisfaction of all group members.

The lesser part of this project was the organizational aspect. We did not have any workspaces, so we were sometimes forced to work separately. This wasn't so bad for the first couple of weeks, because a lot of work could be done separately. However, we all agreed that we should at least work in the same space when we were in the implementation phase. So, during the last weeks, we worked wherever we could find enough place for us all to sit and sometimes we had to move halfway through the day, because the room was reserved. This was very disturbing.

-Marco

At first, working with three students half my age seemed accident-prone. Four people with different backgrounds, cultures, ages and future plans, this was failure waiting to happen. However, it wasn't like that at all. I have learned so much from the other team members, I enjoyed every part of the project. We were particularly good in problem analysis, learning new technologies and delivering just-in-time. I did not like the fact that a place to work as a group was not to be found, so we had to do a lot of work individually, at home. Perhaps if there was a work place, we could have reached more goals. That would also have resulted in better adjustment of the individual tasks. Now, too much time was lost waiting for results of other team members.

With a limited knowledge of Java, I focused on testing the code of others. My project management and documentation skills came in handy.

-Jozua

As stated in the acknowledgements, at first I wasn't quite excited by this project at all. However, because of my experience with Android, Java and application development, as development lead I was right where I needed to be. Some unfortunate circumstances heavily delayed our implementation phase and for that reason I'd like to reflect on the implementation phase. Preparation for the phase was good, proper planning and communicating lead us head strong into a stage of development on a platform none of us feel absolutely comfortable with. Android has solid constraints on its applications that require some unique coding patterns. Because of this, one of the plans we made - which was to divide tasks such that in theory everyone could work solo for the majority of the entire development cycle - in the end may have been a mistake. Integration of components require a good understanding of Android's IPC (inter process

communication) framework. When one component is delayed, our integration was automatically delayed and that may have been our biggest mistake, since we ended up delaying our integration until almost the last week. Of course without integration, it's very difficult to test the workings of the system as a whole, and for developers inexperienced in dealing with such bottom-up approaches it is simply a cause for increased concern, increased delays and gradual loss of control over the entire situation as time goes on. If we had used SCRUM in the way we had planned, it may have improved the flow a little bit, as this would urge our team members to stay aware of the status of other components. More important than staying aware is to keep a tight schedule to plan the integration process. After we had found that our weekly sprints were not feasible, integration became the least of our worries and we focused instead on completing our own components. This choice was obvious because each member feels mostly responsible for their own components, but we failed to remember that integrating components and getting even 10% of our targeted system to work is a major motivation boost. On top of that, building intermediate prototypes is equally important for the client, who can then comment on the system. What I think we should have done, following finding out that our deadlines were too difficult to keep, is simply adjust our deadlines to ensure that integration and prototype development maintain a very high priority

-Randy

At the start of this project was I rather eager to work on an application that needed to be developed for the Android platform. The positive testimonials may have caused some of my eagerness. It is as always a unique experience working with people from different backgrounds it is comforting to know that we all have a common goal.

The lack of workplaces during the early stages of the project was rather worrisome. At the time it didn't seem to be much of an issue, but as time passed it started to get in the way of our progression. Luckily we managed to be resourceful at finding a place to do our work. The design phase went without much trouble, during which I really wanted to start programming. As soon as the implementation phase started it became clear how much I underestimated Android. It was very easy to receive error or get stuck trying to figure out how to implement something. This lead to delays in development that could have been avoided had there been sufficient documentation for the implementation of the calendar component. Testing on the Android device I received from TNO was hampered by the lack of internet access to Eduroam. This resulted in having to ask a sim card from the other teammates in order to continue working on my tasks. Although we have only been able to implement the Must haves and some of the Should haves from the MoSCoW model, we should be happy to have accomplished this much in a short amount of time. It is possible that if we made use of SCRUM in how it is meant to be applied, it might have led us to progress much faster.

Overall it was memorable experience to work on this project and my teammates have been a source of inspiration through these past 10 weeks.

Strengths and Weaknesses

The project group consists of four students who have never worked together at a project before. At the start of the project, the group had a meeting to determine the strengths and weaknesses of each member and the opportunities to gain more experience from this project. This resulted in an formal allocation of the project tasks. However, this was not strict as some roles were switched between group members alternately for a week or two.

- Loubna
Strengths: research, analysing
Weaknesses: programming, brainstorming, risk management, presenting
Opportunities: agile project development, project management, making user manual

- Marco
Strenghts: project management, agile development, presenting, design
Weaknesses: Java programming, testing, research, analysing
Opportunities: English language, testing, reporting

- Jozua
Strenghts: Java programming, databases, testing, research
Weaknesses: planning (time estimation), risk management
Opportunities:presenting, agile project development, communication

- Randy
Strenghts: programming, English language, research
Weaknesses: presenting, reporting, communicating with the customer
Opportunities: agile project development, testing

Appendix B. List of figures

Appendix C. list of tables